Crop Recommendation System

Neha Nagesh Shetty | Rachana Anil | Savitha Munirajaiah
shetty.ne@northeastern.edu ; anil.r@northeastern.edu ; munirajaiah.s@northeastern.edu
Northeastern University, Boston - MA

Abstract

Agriculture has been severely impacted by climate change due to unpredictable weather conditions. Picking the right crop to grow is one of the most critical decision a farmer needs to make to gain maximum yield. Traditional techniques are not very effective due to the current uncertainties involved. However, a modern farming technique called Precision agriculture leverages Internet of Things and Machine Learning techniques to collect real time data and further analyze the data to make better decisions.
Crop recommendation system applies classification techniques using Machine learning algorithms on real time data such as soil parameters, temperature, humidity, etc,. to recommend a crop for farmers to sow, helping them directly. Gaussian Naive Bayes, Neural Networks, Logistic Regression, Support Vector Machine, etc,. are amongst the list of algorithms used to develop models for this crop recommendation system. Performance of these models are compared based on their accuracy level, time and space complexity. The algorithm that provides the best results is chosen for the application.

Agriculture is one of the main source of supply for human needs. It is a primary occupation and is one of the major industrial sectors in many nations. Farmers often follow traditional methods based on naked eye observations while making decisions related to farming. But, due to globalization, increase in population, and climate change a lot of uncertainty is incorporated. Traditional techniques may not consider and cope with such rapid changes and drastic effects. Hence, a more robust technique is necessary where the data about the features affecting farming are collected periodically and are monitored in real time.

Precision agriculture is a modern farming technique which leverages Internet of Things and Machine Learning techniques. It applies IoT methods to collect real time data on several features such as soil parameter, temperature, humidity, etc,. These parameters are collected using sensor nodes and are stored in cloud for monitoring. Several supervised and unsupervised machine learning algorithms can be applied on this data to further analyze and perform predictive analysis to help farmers make informed decisions.

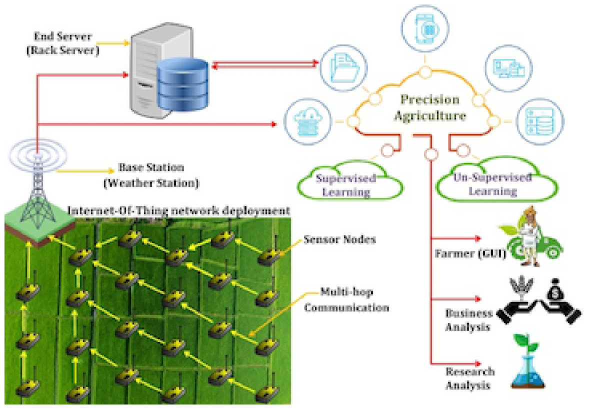


Figure 1: Precision Agriculture

One of the most important decision a farmer makes is to pick the right crop to sow. We can use Precision agriculture techniques discussed above to help with this decision by developing a Crop Recommendation System.

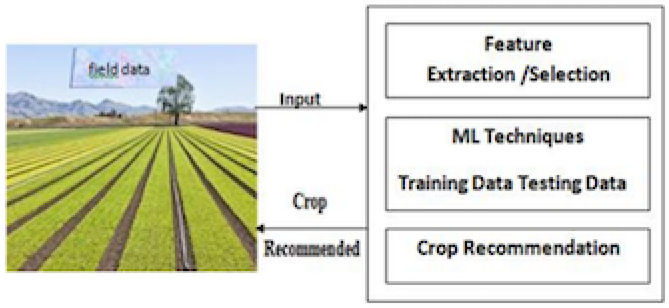


Figure 2: Crop Recommendation System

The Basic idea behind this system is to collect field data from which features are extracted and processed. On this processed data, several Machine learning algorithms are used to develop several prediction models. Comparing these predictive models the best model is used for our Crop recommendation system. This system will then recommend the best crop to sow under the given conditions in real time.

Background

Classification is a supervised machine learning technique that is used to predict discrete values associated with a set of features. In other words, classification is a process of approximating a mapping function(f) such that input variables(X) are mapped to discrete output variables(Y).

$$Input(X) \xrightarrow{Mappingfunction(f)} Output(Y)$$

In classification terminology, input variables are called features and output variables are called classes.

In a general classification problem, the dataset is split into 3 parts - training, validation and testing. The size of the training dataset is larger compared to the other two datasets. The classifier is trained using the training dataset. When one wants to predict the class values, the classifier associates the feature values with the training dataset and makes a predication. The validation data set is used to check how accurately the classifier was trained based on the predictions made. If the accuracy is low, the classifier is trained again with different parameters values and validated again. The testing dataset is run on the classifier with the best validation result and the accuracy is computed.

Let us consider a set of input variables or features.

$$X = \begin{bmatrix} x1 \\ x2 \\ . \\ . \\ xm \end{bmatrix}$$

Each instance of input variables belong one of the n classes denoted as Yi, where i = 1,...,n. The training dataset can be represented as

$$[X_i, Y_j]_{t=1}^{T}$$

where X_i is the instance of inputs and Y_j is one of classes respectively. There are T rows in the training data set.

While training the classifier, we need to learn the boundary separating the instances of one class from the instances of all other classes. We view a n-class classification problem as n two-class problems. The training examples belonging to Y_i are the positive instances of hypothesis h and the examples of all other classes are the negative instances of h. Thus in a n-class problem, we have n hypotheses to learn

$$h(X_i) = \begin{cases} 1 & X_i \in Y_i \\ 0 & X_i \in Y_j, i \neq j \end{cases}$$

For each instance in the validation dataset, the n hypothesis is run for the input variables, and a class predication made. Accuracy is the fraction of instances where the prediction of h matches the actual class values

$$Accuracy = \frac{NumberOfCorrectPredications}{TotalNumberOfPredications}$$

This accuracy gives a measure of how well the classifier is trained. Higher the accuracy values, better is the trained classifier.

The classification task that maps the features to exactly two classes is called binary classification. If the number of classes is more than two, it is called a multi-class classification. A classification data set where the classes are not represented equally is an unbalanced dataset, else it's a balanced dataset.
In this paper we are dealing with a multi-class classification problem with an unbalanced dataset.

Related Work

Precision agriculture uses newer technologies including machine learning and internet of things to increase crop yeild.

UChooBoost supervised learning ensemble algorithm is used for classification problem in precision agriculture [1]. Here UChoo classifier is used a base estimator. [1] uses weighted majority voting to evaluate the performance inorder to obtain extended results. This allows UChooBoost to achieve better performance level.

Regularized Greedy Forest(RGF) is used in [3] for crop selection to maximize net yield rate of crops. This paper proposes crop selection based on seasons. It takes into consideration the sowing period, harvesting period and the time needed for the crop growth to predict the yeild rate per hectare.

Apart from crop recommendation, machine learning techniques are used in crop disease classification as well [2]. [2] uses support vector machine, decision tree and neural nets for the classification process. [2] also compares these techniques.

Given the fertility of the soil, [5] predicts the needed nutrients for the plant to grow. [5] uses back propagation neural network for this. [5] suggests that back propagation neural network performs better than other multivariant regression techniques to predict the required nutrients.

Statistics and machine learning techniques is used to predict rice yield [4]. The climate data is clustered using kernel based clustering algorithm and support vector machine is used for classifying rice data. Impact analysis is done on how rice yield changes with variation in climate parameters.

Data Visualization

Exploratory data analysis is the most important and the first step in any machine learning or data related projects as it determines the properties and underlying concepts present in the data. We have used data analysis to determine the distribution of the features' data points for various target labels. This is a supervised machine learning project, wherein we have used a dataset comprising of details regarding the soil composition and weather conditions of the area and it's corresponding best suited crop as the target variable[4].

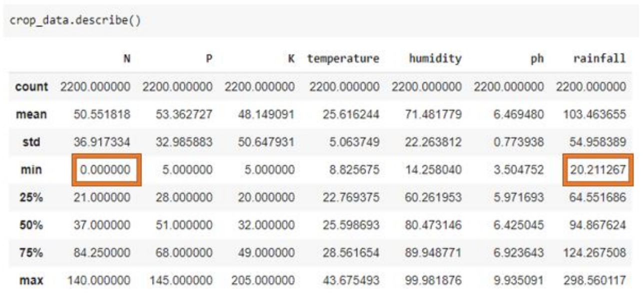
Using the pandas library we read the .csv into a dataframe and using the *read_csv* function. The data consists of 2200 entries with 3 columns of int datatype, 4 of float datatype (7 feature columns) and the target column of object datatype.

Columns description

As previously mentioned, the dataset consists of 2200 rows and 7 columns:

- "**N**" - Entries of integer datatype containing ratio of Nitrogen content in the soil.
- "**P**" - Entries of integer datatype containing ratio of Phosphorus content in the soil.
- "**K**" - Entries of integer datatype containing ratio of Potassium content in the soil.
- "**temperature**" - Entries of float datatype containing temperature in degree Celisius.
- "**humidity**" - Entries of float datatype containing relative humidity in % .
- "**ph**" - Entries of float datatype containing ph value of the soill.
- "**rainfall**" - Entries of float datatype containing rainfall in mm.

The *.describe()* method of the dataframe gives the summary statistics of the dataframe, consisting of minimum, maximum values of each feature as well as the mean and standard deviation. For our data, it was evident that the overall spread of data for each feature is not uniform, as the minimum value of N column was 0 whereas that of rainfall column was 20.211267.

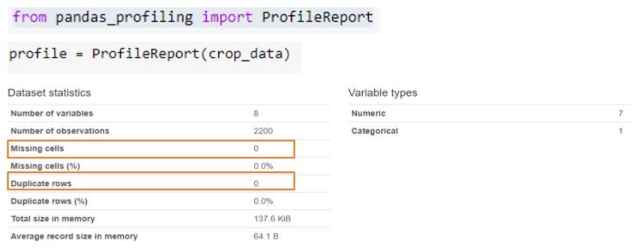


```
crop_data.describe()
```

	N	P	K	temperature	humidity	ph	rainfall
count	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000
mean	50.551818	53.362727	48.149091	25.616244	71.481779	6.469480	103.463655
std	36.917334	32.985883	50.647931	5.063749	22.263812	0.773938	54.958389
min	0.000000	5.000000	5.000000	8.825675	14.258040	3.504752	20.211267
25%	21.000000	28.000000	20.000000	22.769375	60.261953	5.971693	64.551686
50%	37.000000	51.000000	32.000000	25.598693	80.473146	6.425045	94.867624
75%	84.250000	68.000000	49.000000	28.561654	89.948771	6.923643	124.267508
max	140.000000	145.000000	205.000000	43.675493	99.981876	9.935091	298.560117

Figure 3: Summary Statistics of the data

We also used the pandas library to generate the profile report for our data inorder to visualize any shortcomings and determine the null values. From the report, we concluded that the data is clean, i.e there is no null value and all the feature columns had equal number of entries.



```
from pandas_profiling import ProfileReport
```

```
profile = ProfileReport(crop_data)
```

Dataset statistics

Number of variables	8
Number of observations	2200
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	137.6 KiB
Average record size in memory	64.1 B

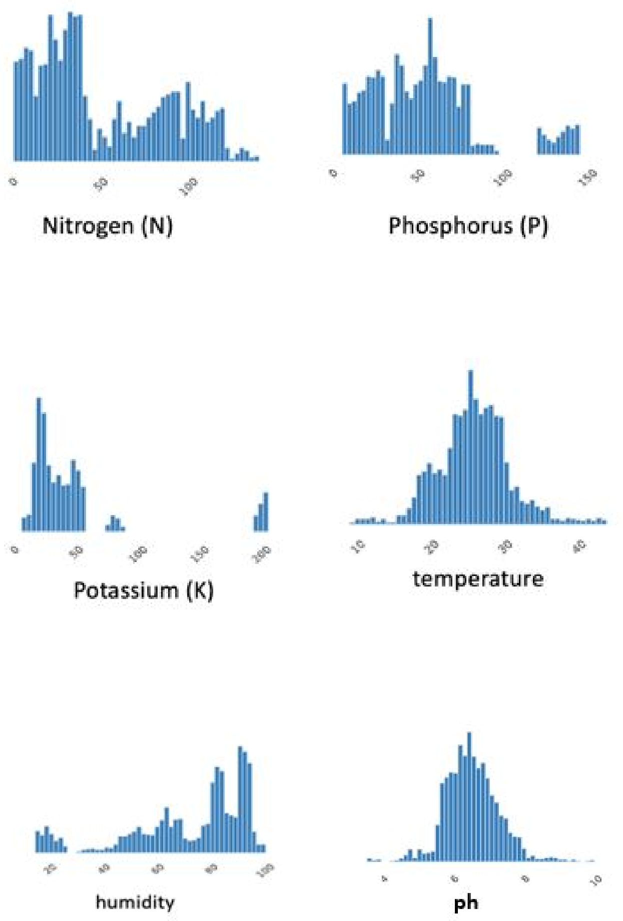
Variable types

Numeric	7
Categorical	1

Figure 4: Profile Report using pandas_profiling

Univariate Analysis

The analysis carried out by taking one of the feature column values with respect to the target column values is called univariate analysis. This analysis gives the visual representation of data spread for each feature value with respect to the target column.



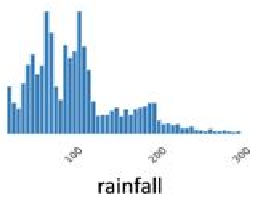


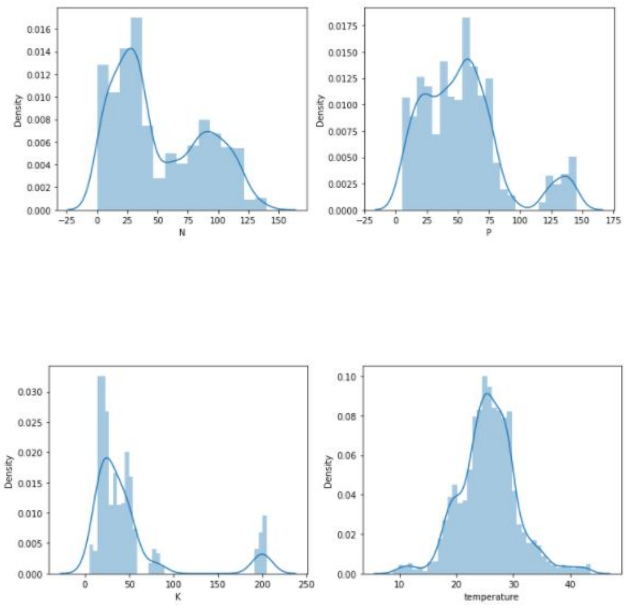
Figure 5: Histogram representation of the feature columns

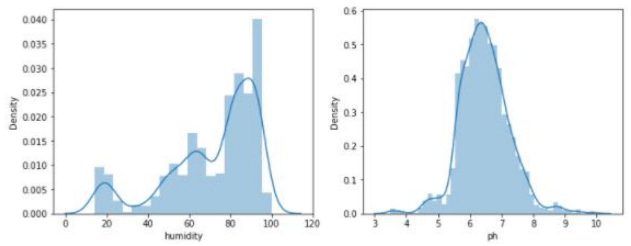
From the univariate distribution of the feature variables, we conclude that the data is not uniformly spread, as already mentioned from the summary statistics.
The data has 22 classes, thus making this a multi-class classification problem. The classes are:

rice	mango
maize	grapes
chickpea	watermelon
kidneybeans	muskmelon
pigeonpeas	apple
mothbeans	orange
mungbean	papaya
blackgram	coconut
lentil	cotton
pomegranate	jute
banana	coffee

Kernel Density Estimation

The kernel density estimation is an important tool for plotting the shape of the distribution. Like the histogram plotted in the univariate analysis, the KDE plot encodes the density of feature values vs the height of the feature values.





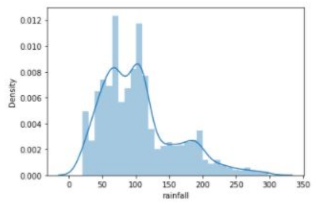


Figure 6: Kernel Density Estimation of the feature columns

From the KDE plots we observed that the histograms are right and left skewed and hence we need to transform the data in order to make it linear.

Boxplot

A box plot, also called as box and whisker plot shows the distribution of quantitative data in a way that is helps to compare variables. The box represents the quartiles in the dataset whereas the whiskers show the rest of the distribution. Using this we can find the features which can be removed.

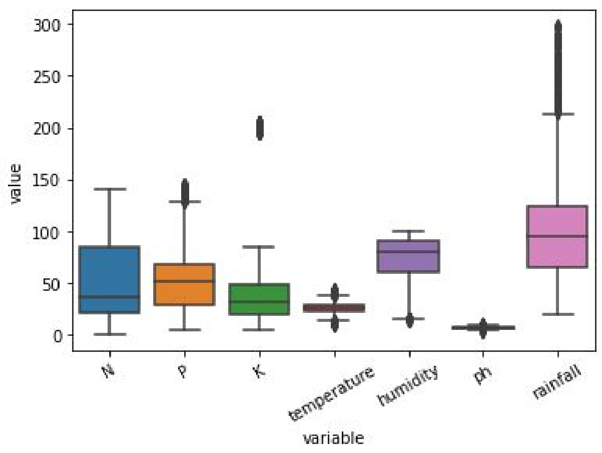


Figure 7: Boxplot representation of the dataset

The box plot is a standardized way of displaying the data based on 5 summary:

- Minimum
- First quartile
- Median

- Third quartile
- Maximum

The rectangular region in a box plot consists of the region from the first quartile to the third quartile(It is also called the inter quartile range). The mid segment in the middle of this region is the median of the feature column and the ends of the whiskers below and above of the box, gives the position of the minimum and maximum values of the datapoint in the respective feature column.
Any point which is 1.5 x IQR(inter quartile range) above the third quartile or 1.5 x IQR below the first quartile, can be classified as outliers. The Inter Quartile Range is calculated as the difference between Third Quartile value and the First Quartile value.
In our dataset, there are few outliers found in the features columns: P , K , temperature, humidity, ph and rainfall.

Bivariate Analysis

In order to determine if any 2 features are related, we used the bivariate analysis. As our feature set consists of all numerical values, this analysis will be Numeric vs Numeric. If any 2 columns are highly correlated, then any one of the column can be dropped as it would not affect the accuracy of the model and hence reduce the complexity of the model. The below figure shows the correlation between the feature columns of our dataset.

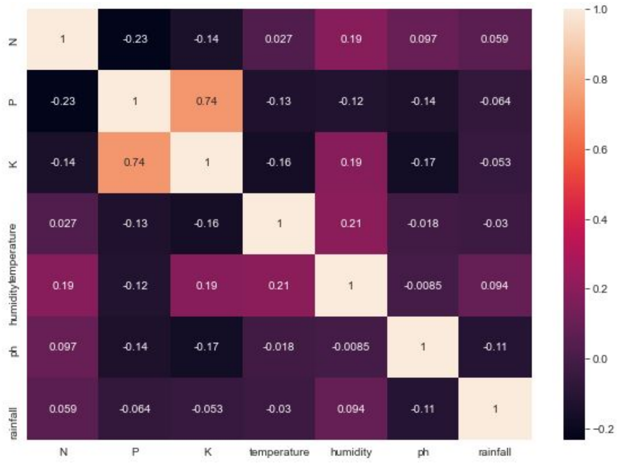


Figure 8: Correlation of the features

From the correlation matrix we observe that columns P and K are highly correlated. But as we have only 7 feature columns, we decided not to drop one of these columns at this stage.

Data Preprocessing

Before we fit our model with our data, we need to clean and trim the data so that the model generates effective results.
From the initial analysis we concluded that our data does not have any null values or missing values. But from the boxplot we found our data to have outliers, which if not removed, would effect the performance of the model.

- **Outliers:** As explained in the Boxplot, any data point which is 1.5 x IQR(inter quartile range) above the third quartile or 1.5 x IQR below the first quartile, can be classified as outliers.The outliers have major effect on the mean of a distribution, as mean of any set of values is given by

  $\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$

  where, N = Total number of observations in the feature column
  x_i = i th observation
  The mean takes every point into account these considering outliers as regular data point. So we need to drop the outliers.

  We have dropped the outliers for each feature based on the quartile values for each column and then replaced this missing outlier values with the median of the column. We replace it with median because our motive is to generalize the data as the data is skewed due to the outliers present.
  Below is the boxplot, kernel density estimation and the corresponding correlation matrix of the features dataset after replacing the outliers present in the data with the median values.

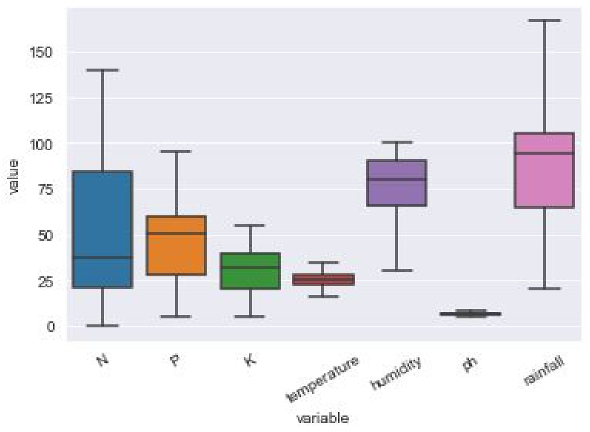
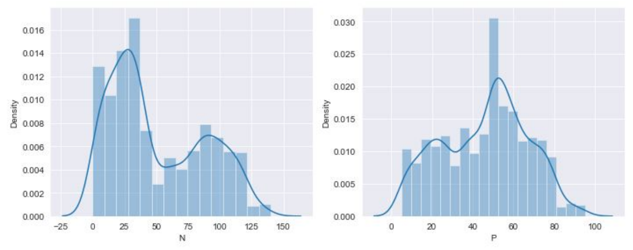


Figure 9: Boxplot representation after removal of outliers



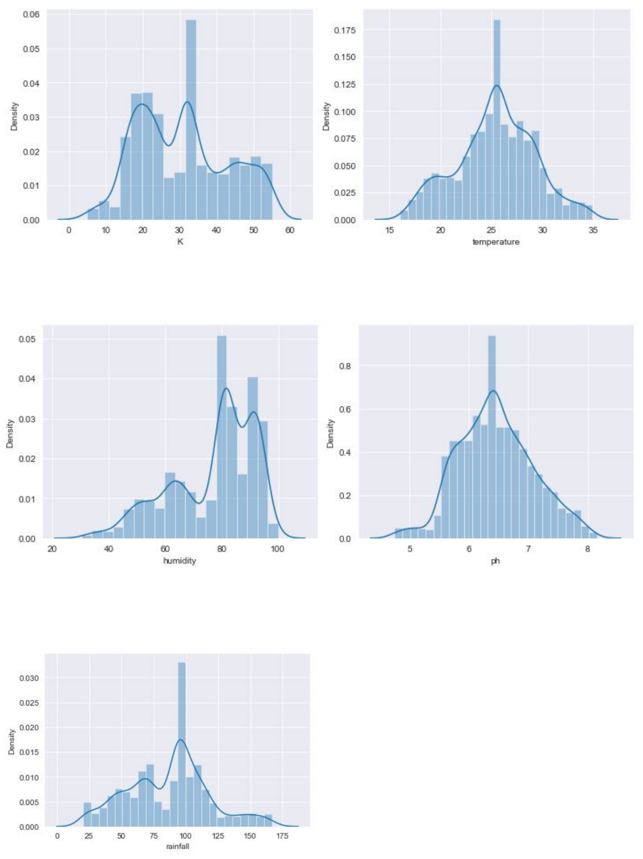


Figure 10: Kernel Density Estimation of the feature columns after removal of outliers

From Figure 8 we can see that after removal of outliers the data distribution is concentrated near the median of each column.

- **Shuffling the data:** We shuffle the data before fitting it in our model in order to have uniform spread of classes in each dataset used for training testing and validation.

- **Splitting the data:** The shuffled data is then split into 60% training data, 20% validation data and 20% test data. This is done using the train_test_split function of sklearn library.
  The data is split into training and testing data in order to make sure that the model is tested on the data it has never seen before. If we test the efficiency of the model on the data we train it, then the model will produce high accuracy as it has already seen the data, and there will be high probability that our model will fail on new data.
  The target column is also separated into training, validation and test sets.

- **Normalize the data:** We normalize our training, validation and testing feature data to a normal scale without distorting the overall range of the features column in order to bring all the datapoints to a similar range.

  We normalize the data using the mean and standard deviation of the feature column, which is calculated as :

  $\frac{x-\mu}{\sigma}$

  where μ= mean of the distribution
  σ = standard deviation calculated as

  $$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

Experiments

Machine learning classification algorithms use input training data to train the classification model. This trained model is used to predict the crop based on the provided features. There are several different classifier algorithms used in our project.

Decision Tree

This algorithm uses a tree structure to classify input features. The tree structure is built on the training data. While classifying the input features, this classifier asks a list of questions regarding the test record being classified. Each time an answer is received, a follow-up question is asked until the class of record is figured out. The questions asked or the order of questions might vary based on the side of tree being traversed by the test record.

In a decision tree classification, the goodness of the split is measured by an impurity measure. A split is said to be pure if all instances of choosing a particular leaf node belong to the same class. When a split is pure, the node doesn't have to be divided further. One of the ways to measure purity of the split is using entropy. Lower the entropy, higher is the purity of node.

Entropy = $\sum_{i=1}^{K} P_i * log_i P$

The other way to measure purity of split is done using gini index. Lower the gini index, better the split.

Gini Index = 1 - $\sum_{i=1}^{K} {P_i}^2$

P_i is the probability of an object being classified to a particular class.
In our implementation we use the the DecisionTreeClassifier model available is the sklearn library to build our random forest Classifier.

The decision tree is tuned using hyper parameter tuning. Several parameter like the impurity measure, depth of tree, minimum number of samples required to split and minimum number of samples on leaf node have been tuned.

Random Forest

Random forest work on the same principle as decision tree. A random forest consists of several decision trees. Each decision tree is constructed using few randomly selected features. The goodness of the split of each tree is measured with impurity measure - gini index or entropy.

While classifying testing samples, each set of feature values are run down all the trees. The class which is chosen by the majority of the trees is assigned to that particular feature list.

In our implementation we use the the RandomForestClassifier model available is the sklearn library to build our random forest Classifier.

Hyper parameter tuning is used to tune the parameters and improve the performance of random forest. Random Forest parametrs that have been tuned are impurity measure, depth of tree, number of trees, minimum number of samples required to split and minimum number of samples on leaf node

AdaBoost

AdaBoost is a learning technique which is used to increase efficiency of base learners. AdaBoost contains a forest of stumps(weak learners). The error produced in each stump influences how the next stumps is being made. The decision of few stumps are given higher weight compared to others.

In AdaBoost, each training sample is provided a weight, $w(i) = \frac{1}{m}$ where m is the number of samples in the training dataset.
Let h_j represents the stumps being used. Here j varies from 0.. number of stumps.
Various stumps are generated by using a set of features. The error produced by a stump while classifying the training dataset is provided by

$$\epsilon = \sum_{i=1}^{m} w_t(i)[y_i \neq h(x_i)]$$

The stump with least classifying error is chosen. The amount of say the chosen stump has in the final decision is

$$\alpha = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$$

Upon stump selection, the weights of the training data set samples are updated based on weather the instance was rightly classified by the chosen stump.
If the training instance is rightly classified, then their weights decrease compared to their previous value

$$w_{t+1}(i) = w_t(i) * e^{-\alpha}$$

If training instance is incorrectly classified, then their weights increase compared to their previous value

$$w_{t+1}(i) = w_t(i) * e^{\alpha}$$

The new stumps are generated based on updated weights of the training samples.
In our implementation we use the the AdaBoost model available is the sklearn library to build our AdaBoost Classifier.

Similar to the previous algorithms, hyper parameter tuning was applied to AdaBoost. SVM, decision tree and random forest were used as base estimators, with various learning rate.

Gradient Boost

Similar to AdaBoost, gradient boost builds trees of fixed size based on the errors generated by the previous trees. However, unlike stumps these trees are larger in depth.
In gradient boost, we start with with one leaf node that predicts the initial value for every instance in the training dataset. This is done by using log(odds) of the class value. Hence in the first step the log of odds predicts the class which has appeared maximum number of times in the training dataset.

$$log(odds) = \log \frac{numOfChosenClass}{\sum AllOtherClasses}$$

To use log of odds in classification, it is converted to probability.

$$P = \frac{e^{log(odds)}}{1+e^{log(odds}}$$

Generally, when the probability(P) is greater than 0.5, all the values are predicted as chosen value.
Pseudo residuals are calculated by taking the difference between predicted and observed values.

$$Residual = Observed - Predicted$$

This residual value is used to build the tree. Since limited number of leaves are used, one leaf can have multiple values. In order to calculate value at each leaf, a transformation is applied

$$\frac{\sum Residual}{\sum (PreviousProbability)(1-PreviousProbability)}$$

Now in order to improve our previous predication, we can add out new tree with a learning rate to it.

$$Prediction = OldLogOfOdds + (Learningrate * NewLogOfOdds)$$

In our implementation we use the the GradientBoost model available is the sklearn library to build our GradientBoost Classifier. Here sklearn library uses Decision tree Regressor as the base estimator. The library provides no means to change it.
However hyper parameter is used to tune other parameters like loss, learning rate and number of estimators.

XGBoost

Generally, XGBoost has one of the best combination of accuracy and processing time. XGBoost performs best for medium size dataset. It uses decision tree as the base learner

with gradient boosting framework.
In XGBoost, similarity score of leaf nodes and the gain is used to determine how to split the nodes. Higher the gain and similarity score, better is the split.

$$Similarity = \frac{\sum Residual^2}{\sum(PrevProbability)(1-Previousprobability)+\lambda}$$

$$Gain = Left_{similarity} + Right_{similarity} - Root_{similarity}$$

Here, lambda is the regularization parameter. A positive value for λ leads to increase in pruning by reducing similarity score.

In XGBoost, the tree is pruned based on the values on gain and tree complexity parameter γ. The tree is pruned further only if the $Gain - \gamma$ value is positive. Hence, the pruning stops only when $Gain - \gamma$ becomes negative or when maximum depth of the tree is reached.

Once the tree is obtained, the output values are calculated for each leaf node.

$$Output = \frac{\sum Residual}{\sum(PrevProbability)(1-Previousprobability)+\lambda}$$

XGBoost builds the next trees based on the previous tree by using the new residual values. The new residual values are computed using log of odds.
Similarly the trees needs to built one after the other either until residual values reduces to a very small value or the maximum number of trees has been reached.

In our implementation, we have used xgboost package from the python library to build our XGBoost classifier. Hyper parameter tuning is done to tune tree depth, base learner, learning rate, gamma and lambda values to enhance the accuracy of our XGBoost classifier.

Gaussian Naive Bayes

Gaussian Naive Bayes is a variant of Naive Bayes that follows Gaussian normal distribution and supports continuous data. Naive Bayes are a group of supervised machine learning classification algorithms based on Bayes theorem.

$$P(C_k|X) = \frac{P(X|C_k)P(C_k)}{P(X)}$$

C_k is a set of k classes in a dataset and X is a set of features. $P(C_k|X)$ is the posterior probability of class Ck given predictor (features). $P(C_k)$ is the prior probability of class. $P(X|C_k)$ is the likelihood, the probability of the predictor given class. P(X) is the prior probability of predictor.

Using the chain rule, the likelihood $P(X|Ck)$ can be computed as follows:

$$P(X|C_k) = P(x_1, x_2, ...x_n|C_k)$$

$$P(X|C_k) = P(x_1|x_2, x_3, ...x_n, C_k)P(x_2|x_3, ...x_n, C_k)....$$
$$...P(x_{n-1}|x_n, C_k)P(x_n|C_k)$$

Naive Bayes classifiers are used for binary and multi-class classification problems. When binary or categorically input values are provided, Naive Bayes method is very efficient. In Naive Bayes, a Naive Bayes classifier assumes that all the features are not co-related. Hence the name 'Naive Bayes'. Applying conditional independence property as all the features in X are independent:

$$P(C_k|x_1, x_2, ...x_n) = \frac{P(x_1|C_k)P(x_2|C_k)...P(x_n|C_k)P(C_k)}{P(x_1)P(x_2)...P(x_n)}$$

$$P(C_k|x_1, x_2, ...x_n) \propto P(C_k)\prod_{i=1}^{n} P(x_i|Ck)$$

Gaussian Naive Bayes supports continuous valued features and models each as conforming to Gaussian (normal) distribution. The likelihood of the features :

$$P(x_i|C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}}\, e^{-\frac{(x_i-\mu_k)^2}{2\sigma_y^2}}$$

In out implementation we use GaussianNB model in the sklearn.naivebayes library to build our Gaussian Naive Bayes Classifier.

K-nearest Neighbor

K-nearest Neighbor is a sample based learning technique, where it holds all past data sample space while predicting target value for new input sample predictor. It is used to classify a data point based on how its neighbors are classified. KNN works by finding the distances between a query and all the examples in the data, selecting the specified number of examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression).

dist (x, x') $\geq \max_{(x", y")\in S_x)} dist(x, x")$

x is a test input and S_x is a set of k nearest neighbors of x. $S_x \subseteq D$ s.t. $|S_x|$ = k and $\forall(x', y') \in D \setminus S_x$. Here, every point in D but not in S_x is at least as far away from x as the furthest point in S_x

A classifier can then be defined as a function h() returning the most common label in S_x:

h(x) = model $(\{y" : (x", y") \in S_x\})$

This classifier applies distance functions such as Euclidean, Manhattan, Minkowski, etc,. to compute distance from new input sample predictor to all training sample predictors and then k nearest (or smallest) distances are selected with corresponding target values. Target value for new sample predictor is weighted sum of the target values of all the k neighbors. The most common choice is Minkowski distance:

dist(x,z) = $(\sum_{r=1}^{d} |x_r - z_r|^p)^{1/p}$

Selection of k is a puzzle of the method, value of k is directly proportional to the prediction instability (i.e. more data sensitive). Smaller value of k means high variance and low bias and higher value of k means vice versa.

In our implementation, we are using KNeighborsClassifier model in the sklearn.neighbors library to build our K-nearest Neighbor Classifier.

Logistic Regression

Logistic Regression is a supervised learning classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous number values, logistic regression transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes.

Sigmoid Function $\sigma(z) = \frac{1}{1+e^{-z}}$

Here, $\sigma(z)$ is the output between 0 and 1, z is the input to the function, and e is the base of natural log. There are three types of binary classification problem: Binary, Multi, and Ordinal. Binary logistic regression classifies the problem into two classes, while multi-class logistic regression can handle muticlass classification problems.
The hypothesis of logistic regression tends to limit the cost function between 0 and 1. Decision boundaries are necessary to map the output to discrete classes and this point is called as a threshold value or tipping point. For multiple classes, the threshold is usually the highest predicted probability.

$0 \leq h_{\theta(x)} \leq 1$

The hypothesis function of logistic regression for binary classification problem is:

$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

The cost function of Logistic regression for θ parameters is defined as:

$J(\theta) = \frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^i) + (1-y^i) \log(1-h_\theta(x^{(i)}))]$

The key point to note is that the cost function penalizes confident and wrong predictions more than it rewards confident and right predictions. Increasing the prediction accuracy (closer to 0 or 1) has diminishing returns on reducing cost due to the logistic nature of the cost function.

In our implementation, we are using LogisticRegression model from sklearn.linear-model library to build our Logistic Regression Classifier.

Stochastic Gradient Descent

Stochastic gradient descent is an optimization algorithm that estimates the error gradient for the current state of the model using examples from the training dataset, it then updates the weights of the model using back-propagation. The amount that the weights are updated during training is referred to as the step size or learning rate. The learning rate controls how quickly the model is adapted to the problem. Smaller learning rates require more training epochs given the smaller changes made to the weights each update, whereas larger learning rates result in rapid changes and require fewer training epochs

$learning\_rate = initial\_learning\_rate * \frac{1}{1+decay*iteration}$

Stochastic gradient descent attempts to find the global minimum by adjusting the configuration after each training point. Instead of decreasing the error, or finding the gradient, for the entire data set, this method merely decreases the error by approximating the gradient for a randomly selected batch which may be as small as a single training sample. The random selection is achieved by randomly shuffling the dataset and working through batches in a stepwise fashion.

Support Vector Machine

Support vector machine is a supervised learning algorithm used in classification and regression problems. It uses decision planes to classify various classes by drawing decision boundary.

- **Hyperplane:** A hyperplane is similar to a plane except that it is in higher dimensions. To classify the class, our model will draw hyperplane as the decision boundary. But to determine which decision boundary is optimal, we use support vectors hence the name of the algorithm.

- **Support Vectors:** As a rule of thumb, we choose the optimal hyperplane to be the one which segregates the classes better. Support vectors are the data points that are closer to the hyperplane and it effects the position and orientation of the hyperplane. Using these data points the SVM model finds the hyperplane which has maximum margin of the classifier. We use the hyperplane with maximum margin distance because this will give greater confidence for future data points which can be classified correctly.

- **Regulariation Parameters:**

  Significance of C
  Parameter C gives control over the soft-margin. The goal of Soft margin is not to make 0 classification errors, but to make as few mistakes as possible.
  The constraint for Normal Margin is defined as : $y_i(w.x_i + b) \geq 1$
  But for cases where the classes are not linearly separable due to outliers or noise, the condition for the optimal hyperplane is relaxed by adding an extra term:

  $y_i(w.x_i + b) \geq 1 - \xi_i$

where ξ_i is known as slack variables. This variable represent the deviation of the examples from the margin. That results in changing the optimization function from

$Minimize_{w,b}$ for $\frac{1}{2}||w||$
to
$Minimize_{w,b}$ for $\frac{1}{2}||w||$ + $C.\sum_{i=1}^{m}\xi_i$

subject to $y_i(w.x_i+b) \geq 1-\xi(\xi > 0)$ (for any i=1,. . . ,n). This is nothing but adding a penalty term(C) and will result in hyperplane with maximum margin and having smallest error possible.

– A small value of C gives a wider margin, at the cost of some misclassifications.
– A large value of C gives same result as a normal margin classifier.

Effect of Gamma
The gamma parameter defines how far the influence of a single training example reaches. If gamma is too large, then the radius of area of influence of the support vectors only includes the support vector itself. Whereas when the gamma is very small, the model is too constrained and cannot capture the complexity or shape of the data.

- **Kernel:** When we are unable to segregate the classes using a hyperplane in our input space, it is possible that there exists a higher space where a hyperplane can segregate the classes. The idea is to transform the input data into higher dimensional space using a function ϕ(x)
The transformation from 2-D to 3-D is fine, but this transformation is computationally expensive when we have large amount of samples. So SVM uses Kernel.
From the transformation formula for normal margin, $Minimize_{w,b}$ for $\frac{1}{2}||w||$,

we get the following equation:

$L(a) : \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m x_n^T x_m$

where α_n, α_m are known as Lagrange multipliers,
y_n and y_m are your target class points,
x_n and x_m are your data points.

Kernel function is defined as the inner product in some expanded feature space. Let us assume x=$[x_1, x_2, x_3]^T$ and y=$[y_1, y_2, y_3]^T$ be two data point in 3-dimensions.If we need to map x and y then the computational complexity will be $O(n^2)$.

$\phi(x) = (x_1^2, x_1x_2, x_1x_3, x_2x_1, x_2^2, x_2x_3, x_3x_1, x_3x_2, x_3^2)^T$

$\phi(y) = (y_1^2, y_1y_2, y_1y_3, y_2y_1, y_2^2, y_2y_3, y_3y_1, y_3y_2, y_3^2)^T$

$\phi(x)^T\phi(y) = \sum_{i,j=1}^{3} x_1 x_j y_i y_j$

However by using the kernel function k(x,y), we reach the exact same result as above within the 3-dimensional space by calculating the dot product of x^T and y. The computational complexity in this case will be O(n).
$k(x, y) = (x^Ty)^2 = (x_1y_1 + x_2y_2 + x_3y_3)^2$

$= \sum_{i,j=1}^{3} x_i x_j y_i y_j$

Hence kernel trick offers us more efficient and less expensive way to transform the data into higher dimensions.
The different kernels are:

– **Linear Kernel:**
It is defined as K(x,y) = x^Ty. This is the simplest kernel and this is how SVM operates by default.

– **Polynomial Kernel:**
It is defined as K(x,y) = $(x^Ty + c)^q$, when q=1, it is same as the linear kernel. With higher value of q the model starts to overfit.

– **Radial Basis Function(RBF):**
This is also called as Gaussian Kernel. It is defined as K(x,y) = $exp(-\gamma||x - y||^2)$, where γ is one of the regularization parameters.

- **Multiclass SVM:**
SVM is basically a binary classifier but it can also be used to classify multiclass classification problem. It uses an approach called One vs All(OVA). In order to classify the 22 classes, the SVM constructs binary classifiers taking 2 classes at a time.
For a given class, the positive examples are all the datapoints in the class, and all other points are considered as negative class. As a result we get one binary classifier for each of the classes.
For every new prediction, we use each of the classifier and return the classifier which returns a positive result.

In our implementation, we have used svm package from the scikit-learn library to build the SVM classifier. Hyperparameter tuning is done to determine the kernel, and the regularization parameter C and γ.

Multi-Layer Perceptron

Multilayer perceptron is a supervised learning algorithm which maps the input to the corresponding output with the help of non-linear functions used in the layers between the input and the output layers called the hidden layers.
The input layer has m nodes, where m=dimension of the input dataset X=$[x_1, x_2, x_3...x_m]$.
Neurons are the building block of a multi-layer perceptron, which takes in weighted input signals and produces an output signal using an activation function.

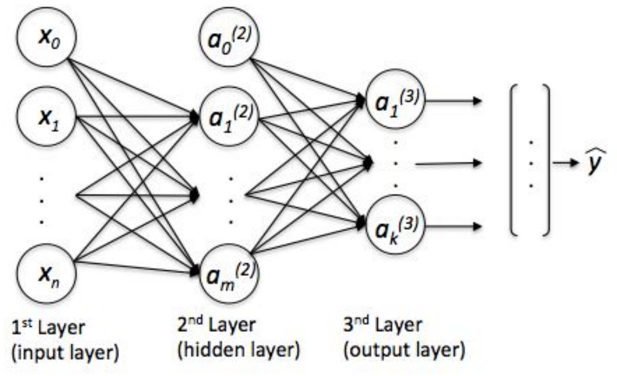


Figure 11: Multi-layer perceptron

From the diagram above, the input data is mapped to the output classed using the hidden layers.The input to these hidden layers is in the form: $a(x) = b + \sum_i w_i x_i = b + w^T x$
where w= connected weight
x = input data point
b = bias

These hidden layers are made up of neuron which use activation function in order to learn the complex patterns in the input data. It decides if the particular neuron is necessary for the computation of the input data based on the weighted input it receives. Some of the activation functions we used in our model are:
ReLU or rectified linear activation function is defined as

$$ReLU(x) = \begin{cases} 1 & \text{if x} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Sigmoid function is a S shaped function in the range [0 , 1].We use this function to map predicted values to probabilities.

Sigmoid Function $S(x) = \frac{1}{1-e^{-x}}$

Softmax function Softmax$(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$

It assigns decimal probabilities to the classes in multiclass classification problem.

In our implementation we use the Sequential model from the keras library, as it is a classification problem and each layer acts as an object that feeds to the next layer. This Sequential model is made up of 2 hidden layers of 50 and 30 neurons respectively and ReLU activation function. The output layer has a softmax activation function. The target column is converted to one hot encoding vector to form a diagonal matrix .The model is compiled using adam optimizer and categorical crossentropy loss is selected for the multi-class classification problem.

Lasso and Ridge Regression

Overfitting occurs when the model follows the training dataset very religiously i.e it gives low training error but may not work very well for generalized data fed to the model during testing. An overfit model is said to have high variance and this will result in poor model performance on unseen data. We use regularization, which reduces the error by fitting the function appropriately and also avoids overfitting of the data. It is used over Regression techniques to improve the accuracy. There are 3 types of regularization:

- **Lasso Regression** also called as L1 Regularization
- **Ridge Regression** also called as L2 Regularization
- **Elastic- net**

We have analysed our data on Lasso and Rigde Regression.

Lasso Regression

LASSO stands for Least Absolute Shrinkage and Selection Operator. The penalty term is sum of absolute values of regression coefficients. The penalty (P) is given by

$$P = \alpha \sum_{i=1}^{n} |\theta_n|$$

Then the cost function after the penalty term can be written as:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta x^(i) - y^(i))^2 + \alpha \sum_{i=1}^{n} |\theta_n|$$

Here θ_0 is not considered because penalizing starts from i=1 to i = n where n is the number of input nodes. The intercept term (θ_0) is not penalized. The α parameter in Lasso tunes the strength of the penalty. Lasso tends to favor model with sparse matrix, i.e a matrix with lot of 0's.Lasso regression can be used for feature selection and feature of less importance can be dropped in case where the feature set size is large.

To determine the value of α for our dataset, we use MultiTaskLassoCV function from the sklearn library. With the optimal α we fit Lasso model on our data to determine the mean squared error and the R^2 score.

Ridge Regression

This works by penalizing the sum of squares of the model coefficients. The penalty (P) is given by

$$P = \alpha \sum_{i=1}^{n} \theta_n^2$$

Then the cost function after the penalty term can be written as:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta x^(i) - y^(i))^2 + \alpha \sum_{i=1}^{n} \theta_n^2$$

The α parameter controls the strength of the penalty, just like in the case of Lasso Regression.

- $\alpha \to 0$, we get the standard linear regression result
- $\alpha \to \infty$, model responses will be suppressed

As we increase the value of alpha, the coefficients approach zero quickly, but in case of Lasso, even at smaller alpha values, the coefficients reduce to zeros. Ridge Regression does not result in elimination of any coefficient.

To determine the value of α for our dataset, we use RidgeCV function from the sklearn library. With the optimal α we fit Ridge model on our data to determine the mean squared error and the R^2 score.

Results

Decision Tree

The graph shows how the accuracy varies with number of features used while classifying.

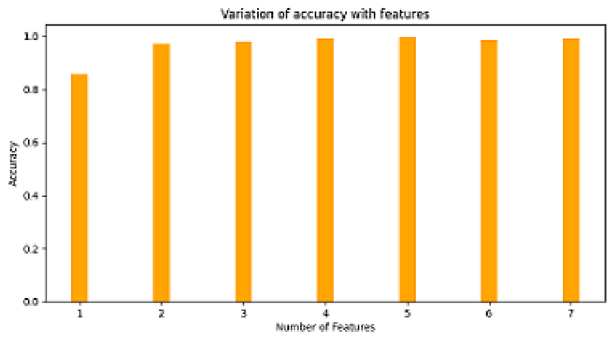


Figure 12: Variation of accuracy with features

Hyper parameter tuning was run on all other parameters as well. The following parameter values provided the best results for decision tree for our dataset.

Impurity measure	Gini Index
Depth of tree	100
Number of features	5
Minimum number of samples on leaf	1
Minimum sample split	5

The accuracy obtained was close to 98.86 percent.

Random Forest

Hyper parameter tuning is done on Random forest classification to find the best fit for our dataset.

The other tuned parameters of the random forest classifier are as mentioned below.

Impurity measure	Entropy
Depth of tree	200
Number of features	3
Minimum number of samples on leaf	5
Minimum sample split	10
Number of estimators	25

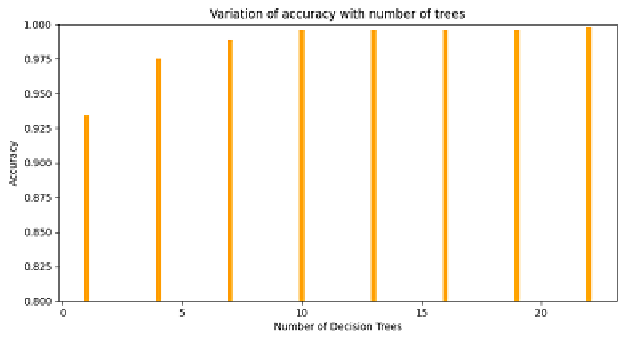


Figure 13: Variation of accuracy with number of decision trees

The accuracy obtained using random forest was close to 99.09 percent.

AdaBoost

The graph indicates how the accuracy of the boosting technique with decision tree as base estimator varies with learning rate.

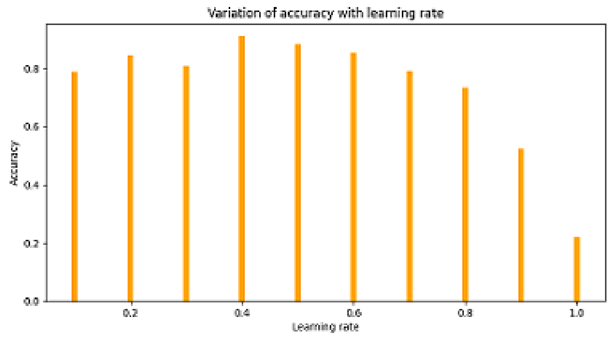


Figure 14: Variation of accuracy with learning rate

It is important to note that these accuracies vary for the same learning rate with different model as base estimator. The tuned parameters of the AdaBoost is mentioned below.

Base estimator	Random Forest
Learning Rate	0.6

The accuracy obtained using random forest was close to 99.24 percent. We can observe that boosting has enhanced the accuracy of it's base estimator.

Gradient Boost

The graph indicates how the accuracy of the boosting technique with decision tree regressor as base estimator varies with depth of trees.

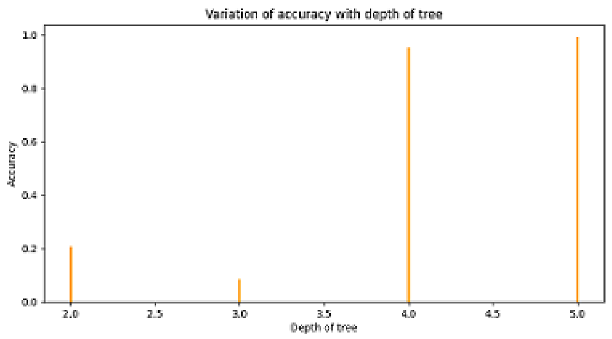


Figure 15: Variation of accuracy with depth of trees

Like in AdaBoost, these accuracies further vary when other parameters are defined/tuned. Here a tree depth of 5 is suitable. However when the other parameters are tuned, tree depth of 5 may or may not be the best suited solution. The best combination of parameters for gradient boost are as mentioned below.

Depth of tree	5
Criterion	friedman mse
Learning rate	1
Number of estimators	100

The accuracy obtained using gradient boost is close to 98.78 percent.

XGBoost

The graph below shows how the accuracy of XGBoost algorith varies with λ values. We know that as λ values increases, pruning increases. However, over pruning a tree is also not efficient as indicated in the figure below. From the figure we can observe that initially the accuracy increases with increase in pruning. However, as we increase lambda further, the accuracies fall due to over pruning.

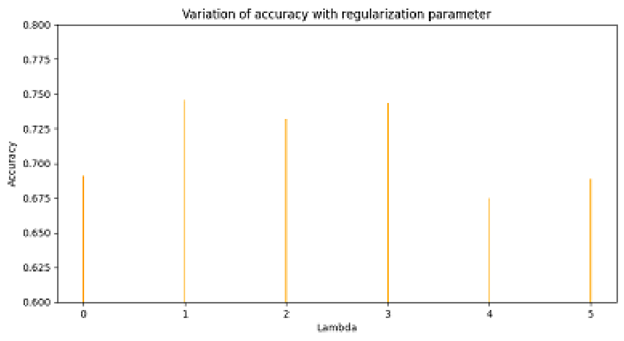


Figure 16: Variation of accuracy with lambda values

Similarly, other parameters of XGBoost were tuned using hyper parameter tuning. The following are the parameter values for which XGBoost provides the highest accuracy.

Depth of tree	8
Booster	dart
gamma	5
Learning rate	0.2
labmda	1

The accuracy obtained using XGBoost is close to 96.5 percent.

Logistic Regression

The graph indicates the impact of parameters on the accuracy of logistic regression model. Here, the accuracy changes as the solver used for the classification varies. Based on the graph it is clear that the model gives the best results when ’newton-cg’ is its solver.

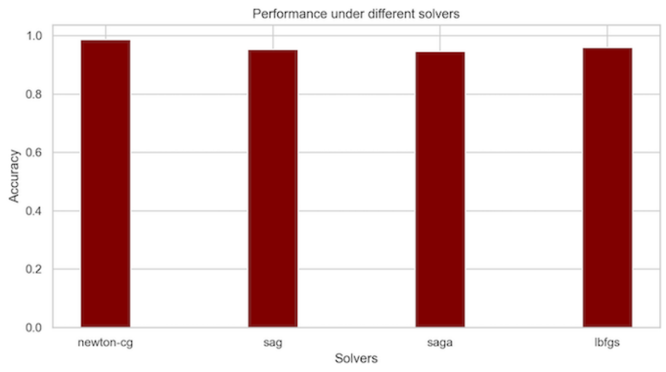


Figure 17: Variation of accuracy with solvers

Hyper parameter tuning is performed on parameters of logistic regression classification to find the best fit for our dataset. The best combination of parameters for our logistic regression model are :

Penalty	l2
Solver	newton-cg
Class weight	balanced

The accuracy obtained using logistic regression is 98.69 percent.

Gaussian Naive Bayes

The graph indicates the impact of the parameters on the accuracy of gaussian naive bayes model. Here, the accuracy changes as the var-smoothing parameter is changed for the classification. The variable var-smoothing artificially adds a user-defined value to the distribution’s variance, essentially widening the curve and accounting for more samples that are further away from the distribution mean.

Hyper parameter tuning is performed on parameters of Gaussian Naive Bayes classifier to find the best fit for our dataset. The best combination of parameters for our gaussian naive bayes model is :

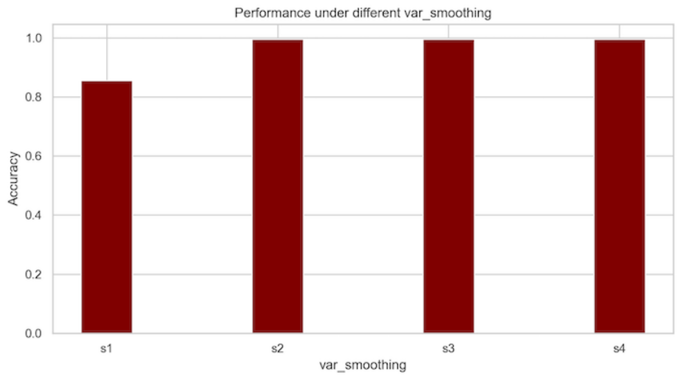


Figure 18: Variation of accuracy with var-smoothing

var_smoothing: 2.848035868435799e-05

The accuracy obtained using Gaussian Naive Bayes is 99.09 percent.

K-nearest Neighbor

The graph indicates the impact of parameters on the accuracy of k-nearest neighbor model. Here, the accuracy changes as the number of neighbors used for the classification varies. Based on the graph it is clear that the model gives the best results when n-neighbor is 10.

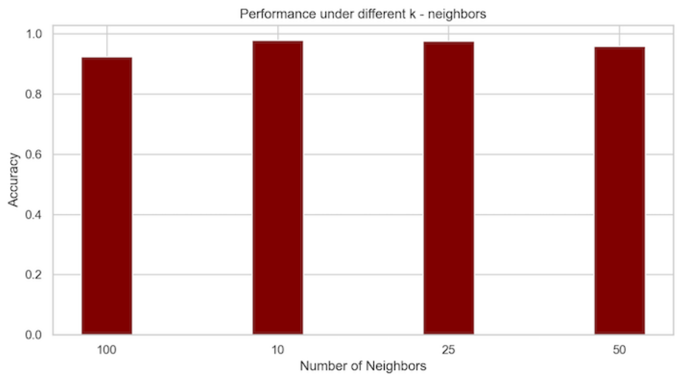


Figure 19: Variation of accuracy with n-neighbors

Hyper parameter tuning is performed on parameters of KNN classification to find the best fit for our dataset. The best combination of parameters for our k-nearest neighbor model are :

N-neighbors	10
Weight	uniform
Metric	manhattan
Algorithm	auto

The accuracy obtained using K-nearest Neighbor is 97.95 percent.

Stochastic Gradient Descent

Stochastic Gradient Descent model is unpredictable and the accuracy of its classification cannot be determined because it varies time. This model will not be the best for our application due to its uncertainty. However, conducting hyperparameter tuning indicates indicates the impact of parameters on the accuracy stochastic gradient model. Here the accuracy is plotted for varying loss function used for classification.

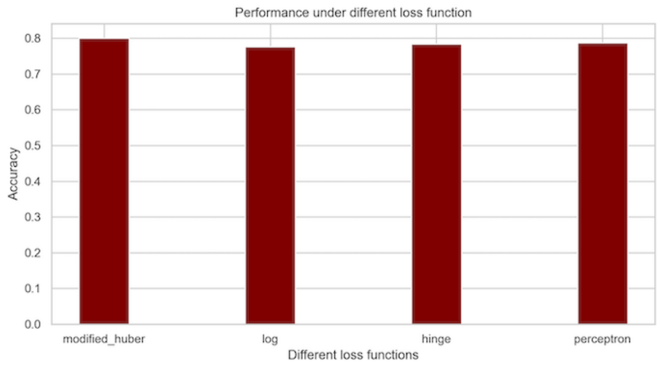


Figure 20: Variation of accuracy with loss functions

Hyper parameter tuning is performed on parameters of stochastic gradient descent classification to find the best fit for our dataset. The best combination of parameters for our stochastic gradient descent model are :

Loss	modified-huber
Penalty	l1
Class weight	balanced
Learning rate	optimal

The accuracy obtained using stochastic gradient descent is 80.00 percent.

Support Vector Machine

Hyperparameter tuning was run on all the parameters to find the best fit parameter for our data. The following parameter values provided the best results for SVM:

C	1
γ	0.1
kernel	rbf

Below is the plot of binary classification(by taking 2 classes, namely apple and banana) vs 2 input features N and P

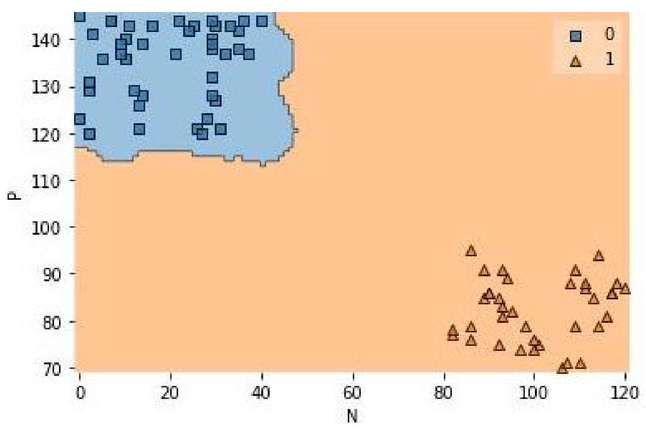


Figure 21: Binary classifiers constructed for OVA approach

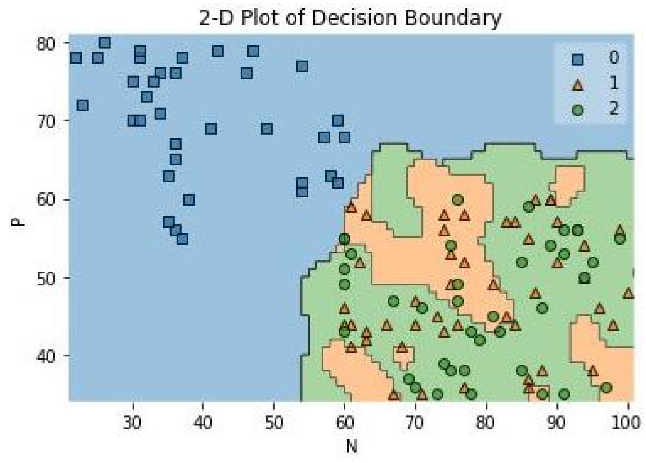


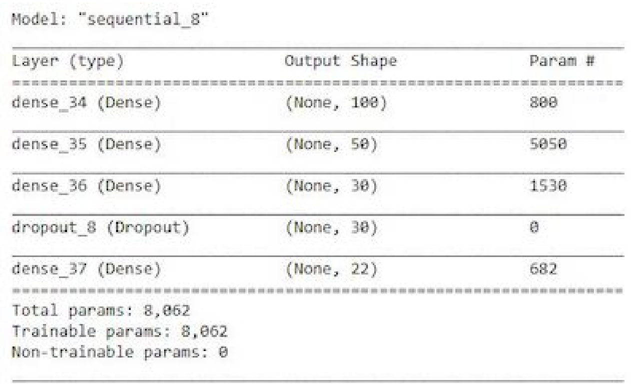
Figure 22: 2-D plot of Decision Boundary

The above 2-D plot is with respect to 3 target classes. The decision boundary for all 22 classes in 2-Dimensions is shown below.

The accuracy obtained using Support Vector Machine was close to 96.59 %

Multi-Layer Perceptron

The figure below shows the Sequential model summary for the multi-layer perceptron, giving details regarding the number of nodes in each layer and shape of the output of each layer.



```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_34 (Dense)             (None, 100)               800
_________________________________________________________________
dense_35 (Dense)             (None, 50)                5050
_________________________________________________________________
dense_36 (Dense)             (None, 30)                1530
_________________________________________________________________
dropout_8 (Dropout)          (None, 30)                0
_________________________________________________________________
dense_37 (Dense)             (None, 22)                682
=================================================================
Total params: 8,062
Trainable params: 8,062
Non-trainable params: 0
_________________________________________________________________
```

Figure 23: Model Summary

We have trained the model with the batch size = 5 for 50 epochs,
where batch size = number of training example in one forward and backward pass(higher the batch size, more the memory you need)
epoch = one forward and backward pass
The below shows the accuracy plot for the same.

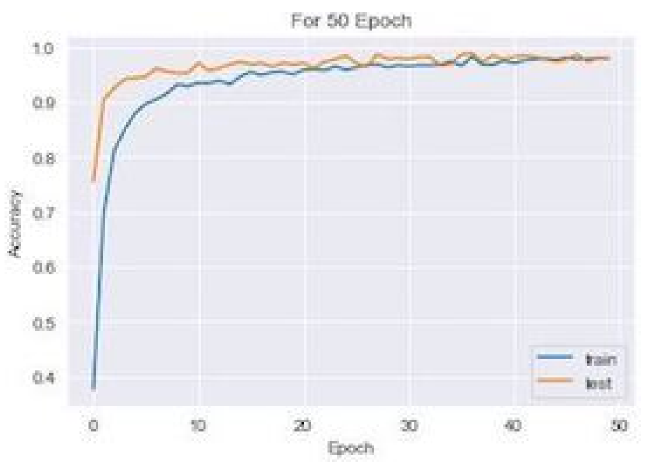


Figure 24: Accuracy graph for 50 epoch

The accuracy obtained was 97.95%

Lasso Regression

The optimum α = 0.1.

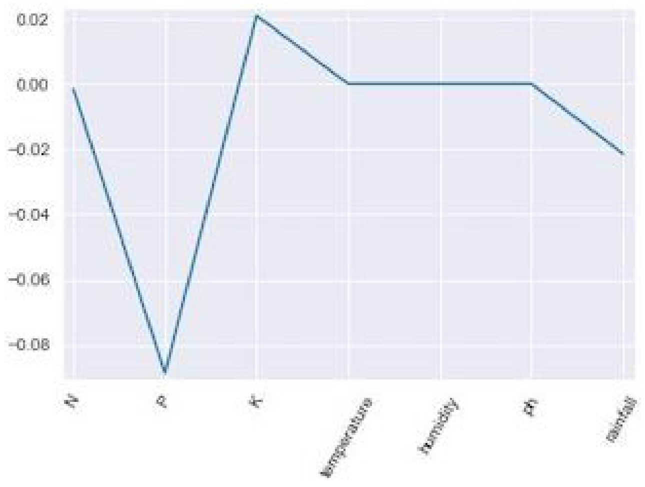


Figure 25: Feature coefficient for α = 0.1

From the graph we can see that feature K is considered to be important feature whereas feature P has less importance. Using this we can fit the data to the Lasso model and determine the Mean-Squared Error and R-Squared Score.

$$\text{Mean-Squared Score} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

where, n = number of datapoints
Y_i = actual value
$\hat{Y}_i$ = predicted value
This gives the average of the squared error.

$$\text{R-Squared Score} = 1 - \frac{RSS}{TSS}$$

where, RSS = sum of squares of residuals
TSS = total sum of squares
This gives how well the model fit on our data. The higher the value the better.
Using the α value, the Lasso model gave a mean squared error of 0.0434 and R-Squared score of -0.00328 which indicates that the model performed poorly with the Lasso Regression.

Ridge Regression

The optimum α = 1.0

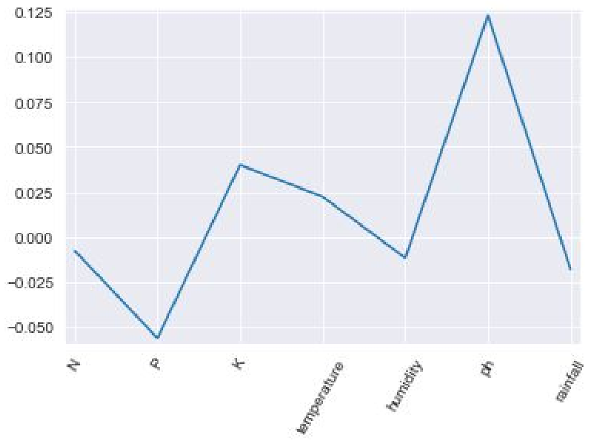


Figure 26: Feature coefficient for α = 1.0

Using the α value, the Rigde model gave a mean squared error of 0.0359 and R-Squared score of 0.1729

Accuracy Comparison

The following graph shows the accuracy produced by various classification techniques for the crop recommendation system.

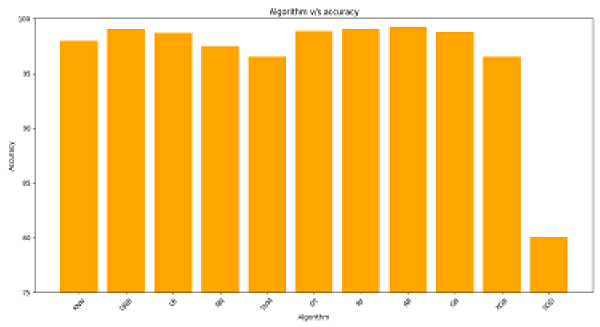


Figure 27: Accuracy of various classification techniques

Most of our algorithms provide an accuracy greater than 95 percent. Stochastic gradient decent provides a low accuracy close to 80 percent and hence not suitable for this classification problem.
Since most of our algorithms work well, the best way to choose the most suiltable algorithm is to compare the space and time complexity of these algorithms.
The bar graphs show the time and space complexity of various classification algorithms used in this project.

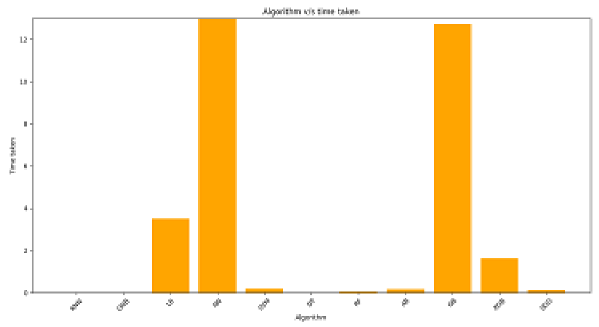


Figure 28: Time complexity of classification algorithms

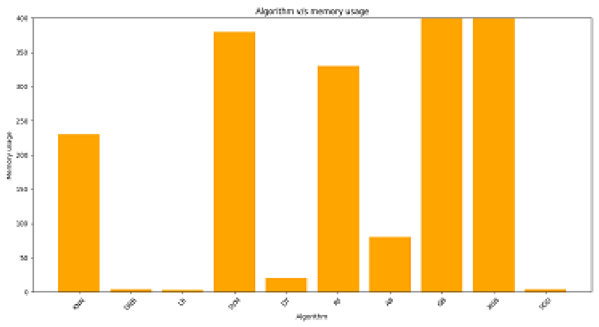


Figure 29: Space complexity of classification algorithms

Though GradientBoost and XGBoost provide an accuracy greater than 95 percent, these algorithms have a very high time complexity. We have other algorithms with similar or better accuracy and smaller runtime. Hence GradientBoost and XGBoost are not a suitable algorithms for our crop recommendation system.

Apart from these most of our algorithms provide an accuracy greater than or close to 97 percent. Hence an algorithm with the top accuracy, least time and space complexity is chosen. From our analysis we choose Gaussian Naive Bayes as the most suited algorithm for crop recommendation system. This algorithm out performs all other algorithms. Gaussian Naive Bayes has an accuracy of 99.90 percent with time complexity of 68ms and space complexity of a mere 4kb.

High Accuracies

As we can see from the results we have high accuracy for almost all our algorithms. Our methods work so well because the dataset used is very clean. There is no missing values or null values in our dataset. If there were any missing or null values, we would have to replace those with the median values. Moreover we have also replace the outliers in the data with the median value of each feature, which has helped to remove any skewness present in the feature column.
With harder dataset, meaning to have a larger dataset with more features then depending on how clean the data is, our model accuracy would also vary. If the dataset is skewed due to large amount of outliers present, then dropping the outliers would reduces the data thus effecting the effect of certain features on the overall model. This would have tuned our classifier parameters differently and would certainly effect the accuracy of the model.
Apart from that one more point to be noted is the fact that our dataset is very small. However we can't be certain on how a larger dataset would affect the accuracy. There are chances of the accuracy decreasing. increasing or remaining the same based on the dataset.
Also, we tried generating data for few of our classes and tested them on some of our models. What we observed was that the accuracies of those models reduced slightly with this new data and the model was not able to classify every input correctly.

Conclusion

A crop recommendation system is developed in which the most suitable crop is predicted based on the soil and weather parameters. This system maps 7 soil and weather parameters (features) to 22 crops (classes).
The recommendation system is designed with various classifiers such as linear models, ensemble models, and neural nets. After removing the outliers and normalizing the feature values, hyperparameter tuning has been applied to each of the classification models to increase the accuracy of crop prediction. Decision tree, Random Forest, and Gaussian Naive Bayes are the best performing algorithms for this crop dataset. Based on time complexity and accuracies we conclude that Gaussian Naive Bayes is the most suited classification algorithm for the recommendation system. Gaussian Naive Bayes provides an accuracy of 99.09 with time complexity of 68ms and space complexity of 4kb. Having a clean dataset has helped us achieve high accuracy.
The proposed work helps the farmer make an informed decision on choosing the right crop based on soil and weather conditions. This ensures maximum yield which in turn improves the livelihood and economy.

References

1. Anastasiya Kolesnikova, Chi-Hwa Song, Won Don Lee, "Applying UChooBoost algorithm in Precision Agriculture". ACM International Conference on Advances in Computing, Communication and Control, Mumbai, India, January 2009

2. P. Revathi, R. Revathi, Dr.M.Hemalatha, "Comparative Study of Knowl- edge in Crop Diseases Using Machine Learning Techniques". Inter- national Journal of Computer Science and Information Technologies (IJCSIT), Vol. 2 (5), 2180-2182, 2011

3. Crop Selection Method to Maximize Crop Yield Rate using Machine Learning Technique Rakesh Kumar , M.P. Singh , Prabhat Kumar and J.P. Singh, Dept. of CSE NIT Patna, India

4. Mohd. Noor Md. Sap, A. Majid Awan, "Development of an Intelligent Prediction Tool for Rice Yield based

on Machine Learning Techniques ”. Jurnal Teknologi Maklumat, Jilid 18, Bil. 2, Disember 2006.

5. Shivnath Ghosh, Santanu Koley, ”Machine Learning for Soil Fertil- ity and Plant Nutrient Management using Back Propagation Neural Networks”. International Journal on Recent and Innovation Trends in Computing and Communication Volume: 2 Issue: 2, 292 297 ISSN: 2321-8169

6. Introduction to Machine Learning, second edition-Ethem Alpaydın

7. blog.paperspace.com/gradient-boosting-for-classification

8. J. Yang, X. Yang and J. Zhang, ”A Parallel Multi-Class Classification Support Vector Machine Based on Sequential Minimal Optimization,” First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS’06), 2006, pp. 443-446, doi: 10.1109/IMSCCS.2006.20.

9. A. Samvelyan, R. Shaptala and G. Kyselov, ”Exploratory data analysis of Kyiv city petitions,” 2020 IEEE 2nd International Conference on System Analysis & Intelligent Computing (SAIC), 2020, pp. 1-4, doi: 10.1109/SAIC51296.2020.9239185.

10. Mayuresh Shilotri, 26-Weeks-of-Data-Science, (2019), GitHub Repository https://github.com/MayureshShilotri/26-Weeks-Of-Data-Science

11.machinelearningmastery.com/neural-networks-crash-course

12. medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important -98a98db0961d

13. scikit-learn.org

14. statquest.org

15. towardsdatascience.com

16.https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02-kNN.html

17.https://heartbeat.fritz.ai/understanding-the-mathematics-behind-naive-bayes-ab6ee85f50d0

18.https://deepai.org/machine-learning-glossary-and-terms/stochastic-gradient-descent

19.https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

20.https://www.kaggle.com/atharvaingle/crop-recommendation-dataset

21.https://www.agsolutionsgroup.com/