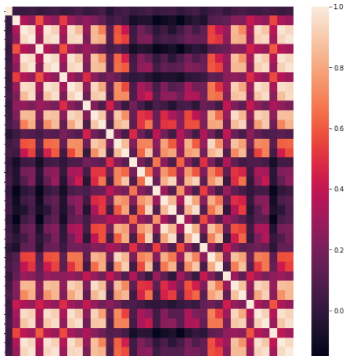
Data exploration

The data were parsed into python as pandas data frames. The data were checked for proper cleaning, making sure that no null values exist, and the data types are identical. The dataframe was searched for duplicates as well and some were found. Using df.describe() from the pandas package allows us to compare the different paraments for each feature such as count, mean and standard deviation. Visualization of the data (38 features including the yield) was achieved using the plot of a heatmap.



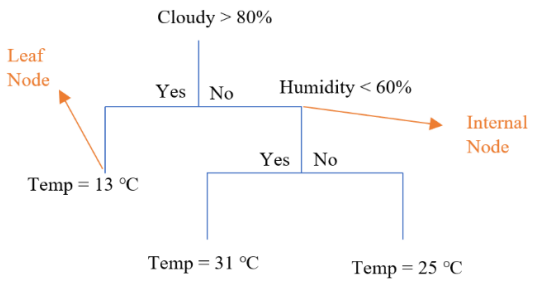
The heatmap shows that there is high correlation between similar seasons no correlation between opposite seasons (Summer, Winter). This can be seen from the big white X on the heatmap. Even though its not clear, there is a strong correlation between the year and the yield implying that throughout the years, the yield increases.

Dataset Preparation

The duplicates were deleted making the total exemplars from 37744 to 31674. All the data, except the desired feature, were standardized (mean = 0, standard deviation = 1). This was done to bring all the variables to common range. Most of the algorithms use arrays as input, except the regression forest, so the data were converted from dataframes to arrays. The data were shuffled to avoid any element of patterns and bias in the dataset. The data was then split for training and testing with a 60:40 ratio, respectively. Special attention was given to the Gaussian Process where the training data was split multiple times (4), in order to decrease the optimization time of the hyperparameters. Furthermore, some important functions needed later for the accuracy of the algorithms were created (mean square error function).

Regression forest

A regression forest uses multiple decision trees the output. This is otherwise known as an ensemble learning method, where an algorithm uses multiple other algorithms to obtain a result with higher precision. Bootstrapping occurs on the data for each tree. Bootstrapping uses random sampling with replacement. This helps to decrease the bias. For regression, inventors recommend p/3 number of random features when building each tree. This is a widely known technique called random subspace. Each tree is made up of many branches, also known as splits. The number of splits is called the depth of the tree. Usually, a preset value is used to avoid overfitting and limit it from growing with unlimited depth. Decision trees have two kind of nodes, internal which contain a split and leaf which contain an answer. Avoiding overfitting can also be achieved by setting a prefix minimum for the leaf size number. The following example shows a simple regression tree predicting the temperature.



The split of each branch can be determined using various techniques. The most popular techniques for regression trees are to minimize variance reduction or to maximize the information gain based on entropy. Both use the variance to the right and to left of the node. This measures how consistent a node is.

Variance of the left node:

$$\sigma_l^2 = E[(f(x) - E[f(x)]^2)]$$

Minimize weighted combination of variance reduction:

$$L_{(split)} = \frac{n_l}{n}\sigma_l^2 + \frac{n_r}{n}\sigma_r^2$$

Maximize information gain of a split using entropy:

$$I_{(split)} = \frac{1}{2}\log(2\pi e\sigma_p^2) - \frac{n_l}{2n}\log(2\pi e\sigma_l^2) - \frac{n_r}{2n}\log(2\pi e\sigma_r^2)$$

$$where\ entropy\ = \frac{1}{2}\log(2\pi e\sigma^2)$$

$$and\ p\ is\ is\ the\ subcript\ for\ the\ parent\ node$$

For the project, variance reduction was used. Finally, regression forest uses multiple decision trees to predict the output using the mean result.

Gaussian Process Regression

Gaussian Processes are distributions over functions *f(x)* of which the distribution is defined by a mean function *m(x)* and positive definite covariance function *k(x,x'):*

$$f(x) \sim GP\left(m(x), k(x, x')\right)$$

$$where\ any\ finite\ subset\ X = \{x_1 \ldots . x_n\}$$

The marginal distribution is a multivariate Gaussian distribution:

$$f(X) \sim GP\left(\mu(X), k(X, X)\right)$$

The general function of a GPR is:

$$y = f(x) + \in_i\ \ where\ \ \in_i\ \sim (0, \sigma^2 I)$$

A kernel is used to describe the covariance *k(x,x')* of the GP random variables. There are many possible kernels like the RBF (radial basis function), periodic or the exponentiated quadratic kernel. In this case the exponentiated quadratic Kernel is used:

$$k(x_a, x_b) = \sigma^2\ exp\left(-\frac{|x_a - x_b|^2}{2l^2}\right)\ where$$

$$where\ \sigma^2 = signal\ variance > 0$$

$$l = lengthscale > 0$$

Using the exponentiated quadratic will result in a smooth prior on functions sampled from the GP. The smoothness is described by the lengthscale, *l.* Therefore, the parameters σ^2 and l must be optimized for maximum accuracy. In this case using gradient descent, which means calculating the derivatives and using them in a cost function until it converges.

Most of the times, the mean function can be assumed to be zero. If that is not the case, a simple transformation will allow the use of mean =0.

The function:

$$f \sim GP\ (\mu, \sigma^2\ )$$

can become:

$$f' \sim GP\ (0, \sigma^2\ )$$

$$using\ the\ transformation\ f' = f - \mu$$

In order to achieve this, the linear regression model can be used. The predictions from the linear regression can be subtracted from the y-train values and form the *f'* for the GPR. Finally, μ and sigma can be used for the GP predictions:

$$P(y_*|X_*, X, y, \theta, \sigma^2\ ) \sim N(\mu, sigma)$$

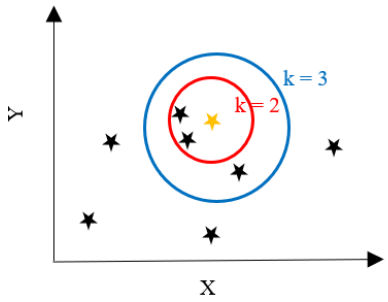
$$where\ \mu = K_{x_*,x}\left(K_{x,x} + \sigma^2 I\right)^{-1} y$$

$$sigma = K_{x_*,x_*} - \left(K_{x_*,x}\right)\left(K_{x,x} + \sigma^2 I\right)^{-1}\left(K_{x,x_*}\right)$$

Additional algorithm and reasoning

The k-nearest neighbors algorithm, knn, was chosen as the last algorithm. The non-parametric attribute of the knn algorithm makes it ideal for the model. It does not assume that the data belongs in any probability distribution. The algorithm is easy to implement, efficient and gives high accuracy results.

The algorithm works by finding the nearest points (neighbors) next to our desired point including noise in the prediction. The number of neighbours, k, is a hyperparameter and the predictions are based on it. The following diagram demonstrates the fundamental idea behind knn, where the yellow star represents the desired point:



The algorithm finds the closest neighbors by calculating the distances from a desired point to every single data point and then chooses the specified k nearest points. A commonly used distance is the Euclidean distance. Lastly, knn averages the respective property values to give an answer.

$$Euclidean\ distance: \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Pros and cons of the algorithms

Regression forests and knn are relatively *computational expensive* processes while GPR is the most computational expensive of all by far. This is a problem as you need more resources to run the algorithms which are not efficient. On the other hand, linear regression is very fast and very efficient.

Furthermore, knn suffers from the *curse of dimensionality*. Knn works very well for a small number of inputs however it struggles at higher dimensions. The rest of the algorithms can operate at high dimensions as well as with lower dimensions.

Moreover, regression forests, GPR and knn are all *non-parametric models*. They do not make assumptions about the data. On the other hand, linear regression assumes that the data are linear and completely fails if they are.

Lastly *accuracy and precision* are important as they determine how close the algorithms are to the true value. Regression forest are very volatile and susceptible to small changes due to high variance. The rest of algorithms have usually a higher accuracy, assuming linearity for the linear regression of course.

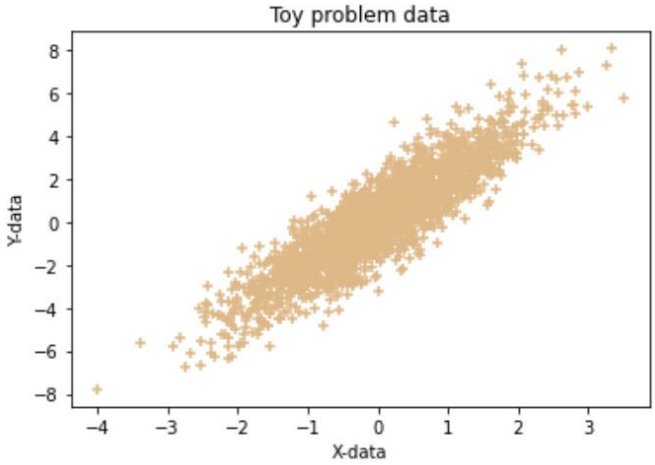
Toy problem design and explanation

The data created for the toy problem were 2000 points based on a normal distribution with mean 0 and standard deviation of 1. The toy problem created is one dimensional, in order to create a simpler model, making the algorithms easier to understand. The function used for creating the y data is the following:

$$y = x * 2 + N(0,1)$$

$$where\ N(0,1)\ is\ some\ noise\ generated\ for\ the\ data$$

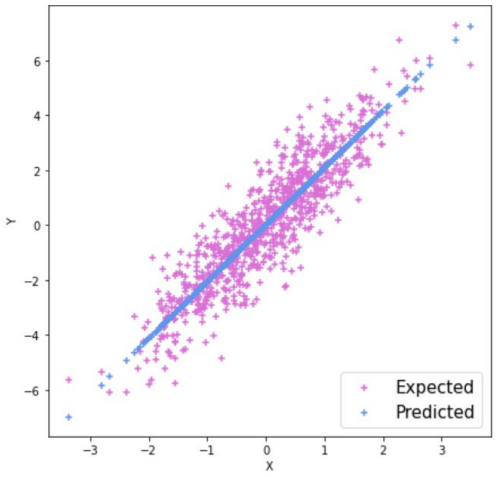
The noise created is for making the relation of x and y harder for the algorithm to find and real life data are not perfectly linear. It is vital in checking the reliability of the algorithms



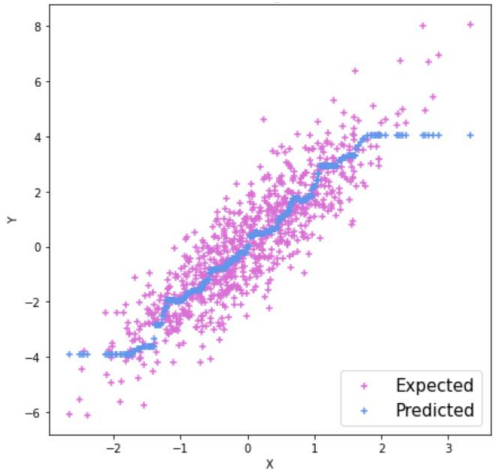
Toy problems results and validation of the algorithms

The following graphs were plotted based on the results of the algorithms:

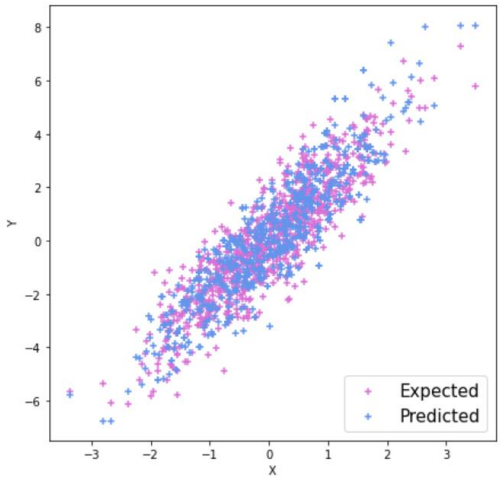
Linear Regression



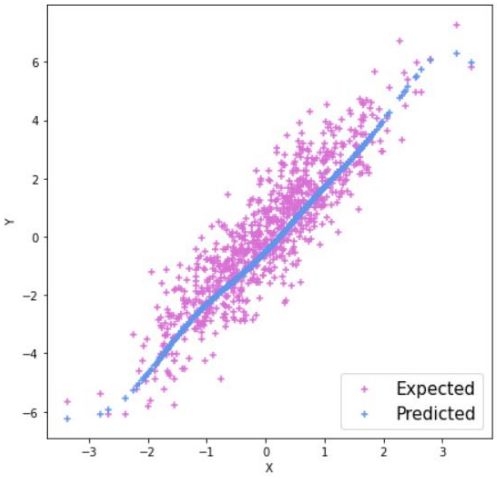
Regression forest



k-nearest neighbors



Gaussian Process



Based on the graphs the linear regression, regression forest and GP filter the noise out of the data and give predictions accordingly. Knn includes the noise in the prediction. Based on the mean squared eror score, mse, the linear regression has the highest accuracy with an mse of 0.979 while the forest follows closely with an mse of 1.027. The GP has an mse of 1.144 while the knn performs the worst with an mse of 1.995. All the mse calculated are relatively low.

This ensures that the all the algorithms are implemented in a proper way and they fit linear data.

Hyperparameters Optimization

For *multivariate linear regression* let:

$$f(x) = w_0x_0 + w_1x_1 \ldots + w_nx_n = \boldsymbol{w}^T\boldsymbol{x}$$

The parameter for the multivariate Linear Regression is the w and it is optimized using the closed form solution:

$$w_* = (X'X)^{-1}X'Y$$

where w_* is the optimized w and X' is the transpose of X

As mentioned before the *regression forest* hyperparameters are the number of trees in each forest, the max depth of each tree and the minimum size of each leaf node. The inventors recommend 5 as the minimum least node size. The code ran many times with different hyperparameters values every time and the mean square error, mse, was calculated to compare the accuracy. Too low mse implies overfitting and high mse implies underfitting. The following results were obtained.

no_of_trees	min_instances	max_depth	MSE
2	5	20	**17.06**
5	5	20	**19.49**
10	5	20	**18.31**
15	5	20	**18.55**
20	5	20	**17.18**
15	1	20	**17.57**
15	5	20	**16.03**
15	10	20	**16.14**
15	20	20	**17.95**
15	30	20	**17.61**
15	5	2	**15.96**
15	5	5	**16.92**
15	5	10	**17.44**
15	5	20	**17.55**
15	5	30	**18.61**

The number of trees selected was 15, the size of the minimum leaf node 5 and a max depth of 20. The hyperparameters chosen balance overfitting and underfitting.

For *GP* hyperparameter optimization occurs when the kernel is optimized. This implies that *lengthscale* and the *signal variance* must be optimized from the train data. The derivatives of the cost function will be used until the cost function converges to an answer. Since the GP is probability based the cost function interval is [0,1]. However, a logarithmic gradient ascent approach will be used transforming the interval to [-∞,0]. Closer to 0 implies overfitting and a very negative number means underfitting. In order to save computation time, the training data were split into 4 groups and their results were combined.

The derivatives used:

$$\frac{d\boldsymbol{L}}{d\log\sigma^2} = -\frac{1}{2}Tr(\boldsymbol{R}^{-1}\sigma^2\mathbf{I}) + \frac{1}{2}Tr(y^T\boldsymbol{R}^{-1}\sigma^2\boldsymbol{I}\boldsymbol{R}^{-1}y) - \frac{1}{2}\log(\sigma^2)$$

$$\frac{d\boldsymbol{L}}{d\log s} = -\frac{1}{2}Tr(\boldsymbol{R}^{-1}\boldsymbol{K}) + \frac{1}{2}Tr(y^T\boldsymbol{R}^{-1}\boldsymbol{K}\boldsymbol{R}^{-1}y) - \frac{1}{2}\log(s)$$

$$\frac{d\boldsymbol{L}}{d\log l} = -\frac{1}{2}Tr\left(\boldsymbol{R}^{-1}\frac{\boldsymbol{d}}{l}\boldsymbol{K}\right) + \frac{1}{2}Tr\left(y^T\boldsymbol{R}^{-1}\frac{\boldsymbol{d}}{l}\boldsymbol{K}\boldsymbol{R}^{-1}y\right) - \frac{1}{2}\log(l)$$

where $\boldsymbol{R} = \boldsymbol{K} + \sigma^2\boldsymbol{I}$

and the last term is the prior (assumed to be zero here)

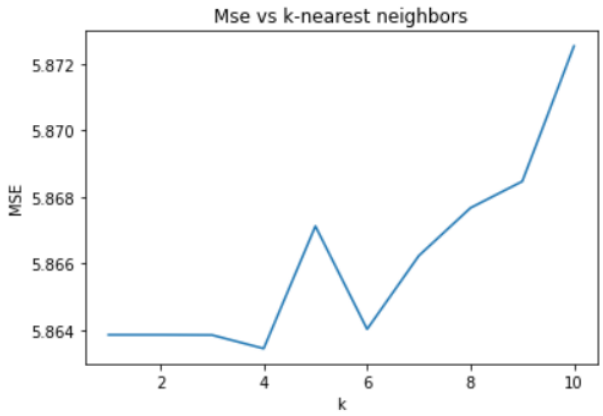
In order to continue with gradient ascent another hyperparameter can be considered, the learning rate, otherwise known as alpha. Alpha will help the cost function to converge as fast as possible. If alpha is too big the cost function overshoots and the answer does not converge. If alpha is too small, it takes many iterations to converge the answer.

alpha	Iterations
1	Overshoots
0.1	Overshoots
0.01	Overshoots
0.001	≈20
0.0001	>100
0.00001	>1000

After running the codes many times, alpha is decided to be 0.001 since the answer converges fast and does not introduce new problems to the algorithm.

For *k-nearest neighbors* the code was run multiple times and the following results were recorded:

k-value	MSE
1	5.8638
2	5.8638
3	5.8638
4	5.8634
5	5.8671
6	5.8640
7	5.8662
8	5.8677
9	5.8685
10	5.8725



The jump spike from 4 nearest neighbors to 5 implies that the best k for the knn algorithm to avoid underfitting is below 5. However, choosing 1 or 2 or 3 can be overfitting. Therefore, a k-nearest of 4 was chosen.

Results from the algorithms

The mse obtained for each algorithm are shown the table below:

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

$$where\ n = number\ of\ data\ values, Y_i\ are\ the\ observed\ values, \hat{Y}_i\ are\ the\ predicted\ values$$

Algorithm	MSE
Linear Regression	9.00
Regression forest	17.55
Gaussian Process	5.84
K-nearest neighbors	5.86

Gaussian process scores the highest accuracy with the smallest mse and knn is really close in accuracy with an mse of 5.86. On the other hand, the Regression forest scores the lowest accuracy with the highest mse. A relatively low mse for the linear regression implies some linearity in the data. Knn having on of the best accuracies concludes that there is a lot of noise in the data.

Complexity and optimal complexity of each algorithm

The runtime of each algorithm is shown in the table below:

Algorithm	Avg training time/s	Avg prediction time/s	Avg total time/s	Ratio
Linear Regression	0.007	0.001	0.008	1 (10^0)
Regression forest	82.082	0.014	82.096	≈10000 (≈10^4)
Gaussian Process Regression	777.893	553.520	1331.410	≈166000 (≈$10^{5.2}$)
k-nearest neighbors	-	39.190	39.190	≈5000 (≈$10^{3.7}$)

From a theoretical point of view the computational complexity of the algorithms are the following where n is the training samples (19074), and p is the number of features (37):

Algorithm	Training	Prediction
Linear Regression	$O(p^2n + p^3)$	$O(p)$
Regression forest	$O(n^2pn_{trees})$	$O(pn_{trees})$
Gaussian Process Regression	$O(n^2pn_{trees} + n^3)$	$O(n_{kernel}p)$
k-nearest neighbors	-	$O(np)$

The computation time of each algorithm depends on its complexity. Gaussian Process is computationally expensive requiring lots of time and memory to run and therefore it does take the most time to run. On the other hand, closed form linear regression requires just one calculation, so it is very fast. The computational time of a regression forest depends on the number of trees and features used. Knn is a lazy algorithm and does not require training since it only predicts values (no model required).

Choosing the right algorithm with reasoning.

Linear regression is not a good model since it assumes linearity of the data and the results clearly show that there is simply not enough linear relationship for the algorithm to accurate. Even though simplistic models are preferred over complex models, Linear regression is far too simple to be used. Therefore, it is rejected.

Looking at the results, Regression forest score the highest mse which means that the accuracy is simply not there. Even though it has some advantages in not needing normalization/standardization of the data, the algorithm is slow, inaccurate and volatile to outliers. It can become very complex as well. Therefore, it is rejected.

On paper Gaussian Process Regression seems to be optimal for predicting with the best accuracy. It performed excellent both with toy problem and the actual data. It seems to filter out the noise from data. However, it is very complex and computationally expensive. This is a huge drawback. It requires far too much time and memory for the algorithm. Therefore, it is rejected.

Knn is the better algorithm for this research. It does not have the drawbacks that the rest of the algorithms have. Knn is relatively fast, easy to optimize and with an easy implementation. Furthermore, no assumptions need to be made about the data and there is no training step. Lastly, sometimes is good to include noise in the predictions since noise exists in real life.

Improving the best performing algorithm.

The Euclidean distance is not the only metric distance that can be used in knn. Some other popular distance metrics are the Minkowski distance and the Manhattan distance. Both have the potential of providing a more accurate estimate for the distance between multi-dimensional variables.

$$Minkowski\ distance = \left(\sum |x_i - y_i|^{1/p}\right)^p$$

$$where\ the\ p\ value\ can\ be\ manipulated\ to\ give\ other\ distances\ (1 = Manhattam, 2 = Euclidean)$$

The Euclidean distance used can be an arbitrary option for many datasets. Using feature weighting the knn algorithm can be drastically improved. Furthermore, knn suffers from the curse of dimensionality. Dimension reduction can help to improve the algorithm's accuracy. Some techniques that combine both feature extraction and dimension reduction are principal component analysis (PCA), canonical correlation analysis (CCA) or linear discriminant analysis (LDA). Then it can be supported by clustering by knn in reduced-dimension space. This will help to remove the noise from data, which is a huge drawback for the knn.

Alternative algorithm

Multicollinearity is a phenomenon where two variables, or in this case features, are highly linearly related, especially when all the features but one, have to do with rainfall. Multicollinearity affects the accuracy of the algorithms. It commonly occurs in models with many features, such as the one provided. Using a special case of Tikhonov regularization called Ridge Regression, is a remarkably effective way to mitigate the problem of multicollinearity.

Ridge Regression avoids overfitting of the data does not require any unbiased estimator. However, Ridge regression algorithms trade variance for bias. This may not be a problem since they add just enough bias to make the estimates relatively reliable approximations to actual population values.

One assumption of the Ridge Regression is the linearity of the data. Even though it was decided from the results of the linear regression that the data are not perfectly linear, it is another way to check that the loss of accuracy using the Linear regression does not arise from multicollinearity and full proof the results of the algorithms.

Discussing if the results are good enough for real world use.

Some algorithms performed well, and their data can be characterized as reliable. Knn even includes some noise which there is a very high possibility of noise in real life as well. However, the predicted results are simply not accurate enough for real world use. Even though the mse for knn turned out to be 5.86, which is relatively accurate, it is not enough for a farmer to trust with the prediction of his yield. Furthermore, the data are only based on the rainfall throughout the year which makes the algorithms biased. There are other external factors that can affect the yield. There is a big chance that the algorithms prediction is wrong, and the farmer will not risk his work based on the estimate of an algorithm. Lastly humanity and human judgement are lost using the algorithm. An experienced farmer should predict the yield better.

Possible breaking points of the solution

Even with dimension reduction, feature extraction and the use of a different metric between neighbors, the knn algorithm can still fail.

The relation of the yield to many dimensions can be vital which implies that many dimensions cannot be removed. This results in a higher dimension problem, making the distances less representative again, introducing the curse of dimensionality one more time. The techniques mentioned above (PCA, CCA, LDA) are complex techniques and require a lot of understanding, prior knowledge and computation time to process. Thus, making the algorithm a complicated one.

Even though other metric distances can be used, they cannot ensure that the predictions will be better. In most of the cases, other metric distances will produce worst predictions than the Euclidean. That is why the Euclidean distance is one of the most popular distances used for the k nearest neighbor algorithm.

Improvements to the dataset

The dataset can be improved by adding new features that are relevant to the yield such as the location of the farm, altitude, fertility of the soil and the personnel size working to produce the yield. This will improve the reliability of the algorithm since it will not be based on just one feature, the rainfall. Another way to improve the dataset is by feature mixing to reduce dimensionality. For example, the minimum, maximum rainfall of a particular month can be reduced to just the average rainfall. The desired outcome can also be split into categories such as wool, meat and milk produced. This is because some of the yield categories are not affected by the rainfall (wool). This can be an external factor that reduces the accuracy of the algorithms. Lastly the data can be normalized, instead of standardized. These improvements can help to increase the accuracy and reliability of the algorithms.