Machine Learning 1 – Project Report

1. What exploration of the data set was conducted?

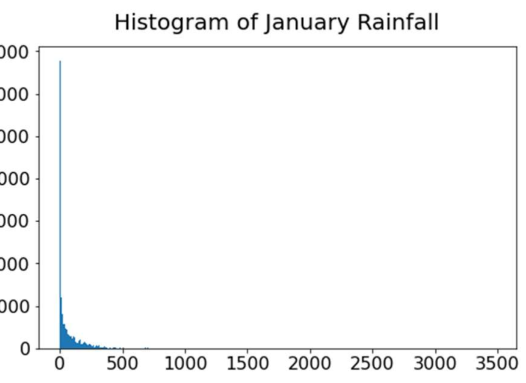
To explore the data, I started by tabulating the minimum and maximum values for each feature. This provided an insight into the features' ranges and allowed me to determine whether their extremes were reasonable. From this, I discovered that the features' ranges varied between orders of magnitude (*Figure 1*), highlighting a need for normalisation to reduce the algorithmic bias this could result in.

	Year	**January Rainfall**	**January Minimum Temperature**
Minimum	1983	0.0	-41.2
Maximum	2013	3476.0	32.4

Figure 1. *Example of wide range of feature scales, showing the minimum and maximum for 3 features.*

By comparing the minimum/maximum temperatures and rainfalls to corresponding world record measurements, I was able to identify some unnatural measurements that were likely due to error.

To provide an insight into the shape of each feature's distribution, their mean and skew were also calculated. This showed that all rainfall features were highly positively skewed, hinting at large outliers (see *Figure 2*). Further to this, I created boxplots and histograms, comparing typical rainfall and temperature distributions. Rainfall features proved more skewed and outlier-prone than temperatures, which tended to have negatively skewed bimodal distributions (*Figure 3*) - probably due to opposite seasons in northern/southern hemispheres).



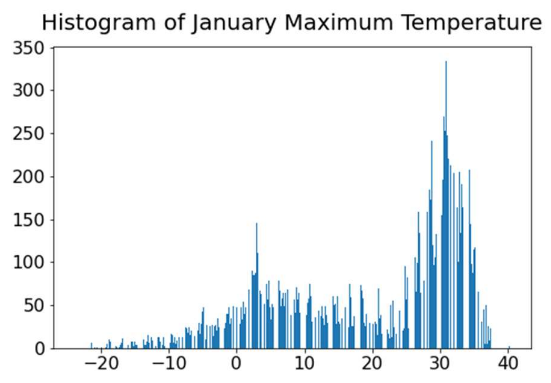
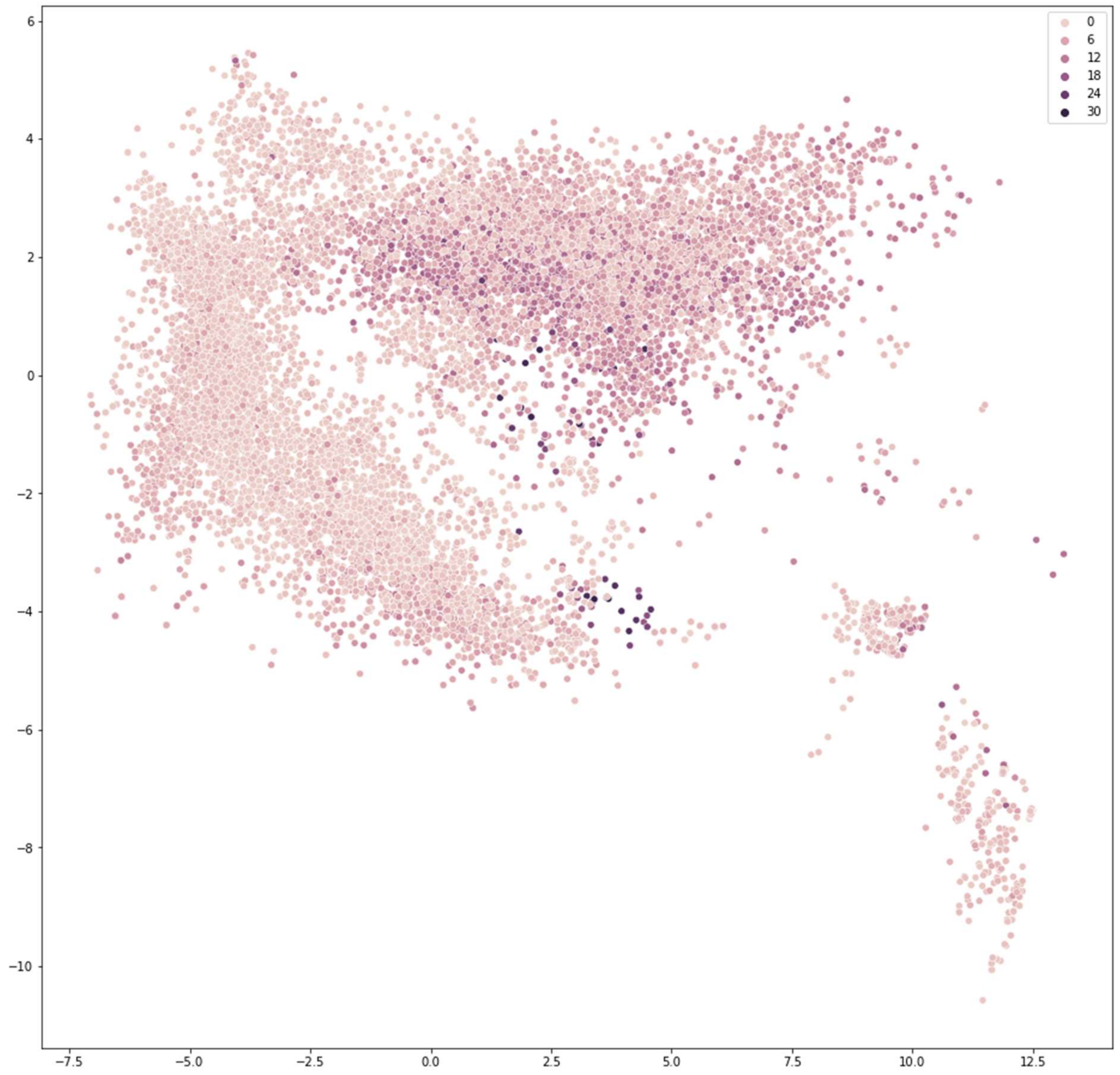


Figure 2 (left). *Histogram showing the distribution of January rainfall feature. This is typical of all rainfall features, with a strong positive skew and large outliers.* ***Figure 3 (right).*** *Histogram showing the distribution of January maximum temperature feature. This is typical of most temperature features, a bimodal distribution with negative skew.*

Additionally, PCA was used to reduce the data to a 2D scatterplot (*Figure 4*), which highlighted a clustered, non-linear structure.

Figure 4. *2D scatterplot of dataset after removal of outliers and normalisation, using PCA dimensionality reduction. Hue indicates the corresponding crop yield. This shows a clustered structure with the darker area indicating a trend for higher yields within that region.*



2. How was the dataset prepared?

For efficient manipulation, the dataset was first read into a NumPy array.

Because outliers are unpredictable by nature, it is reasonable not to expect them to be predicted by the algorithms, therefore, I chose to remove all observations with measurements above 10 standard deviations from a feature's mean, as well as those which had unnatural values (as mentioned above). Combined with the subsequent normalisation of features, this resulted in much closer scales for their interquartile ranges, helping to reduce any algorithmic bias towards features with larger scales (e.g. rainfall). In doing so, I was also able to convert the year (which I assumed to be a continuous variable) to the same scale.

In addition, to reduce the number of features, and hence algorithm runtimes and the likelihood of overfitting, I used dimensionality reduction with PCA. This reduced the dataset to 16 features (a manageable dimension for the algorithms) whilst maintaining 97.9% of its total energy.

To maximise the training data whilst also maintaining enough test observations, I decided to use a 70:15:15 Train:Test:Validation split. I chose to use equal-sized test and validation sets because the validation set is for hyperparameter optimisation and the optimal hyperparameters can change between dataset of different sizes.

3. How does a regression forest work?

In general, a decision tree (DT) works by dividing training data into smaller and smaller daughter nodes (DN) until certain stopping criteria are reached, at which point the new DN become leaves (see *Figure 5*) that predict the average training output within that subset (for regression trees, the mean output).

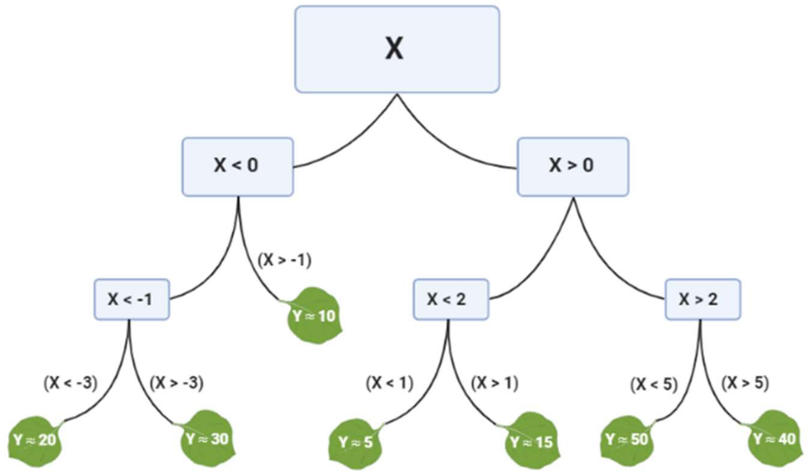


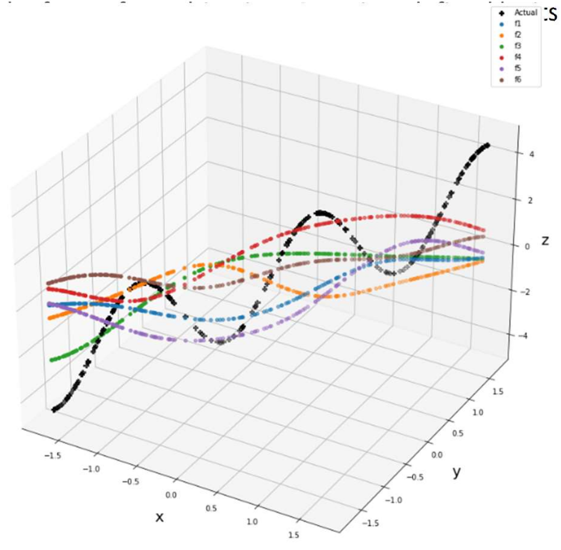
Figure 5. *A typical regression decision tree. X is the data (1D in this example), and each box is a decision node where the best split is determined (best splits are shown in each box).*

Each split into DN is determined by iterating through all possible splits in observations across all features, to find the "best" feature and observation to split at. For a regression tree, the "best" split is usually that which minimises the overall weighted variance of the target variable in the DN (variance reduction), hence increasing their overall homogeneity.

To create a regression forest, we simply generate an ensemble of regression trees in parallel, usually applying "bootstrap aggregation", and take the average of their predictions. Bootstrap aggregation is the process of randomly selecting a set of n training observations with replacement for each tree in the ensemble, where n is total size of the training set. This added randomness increases the overall prediction accuracy of the RF. Additionally, most RF's use a feature subspace method, where only a random subset of features is iterated through at each node. Not only does this reduce the computation required at each node, the randomness also further increases prediction accuracy of the RF.

4. How does a Gaussian process work?

Given some previously unseen input data, with a set of unknown outputs, GP's work by creating an initial prior distribution over the set of possible predictive functions (*Figure 6*). This distribution takes



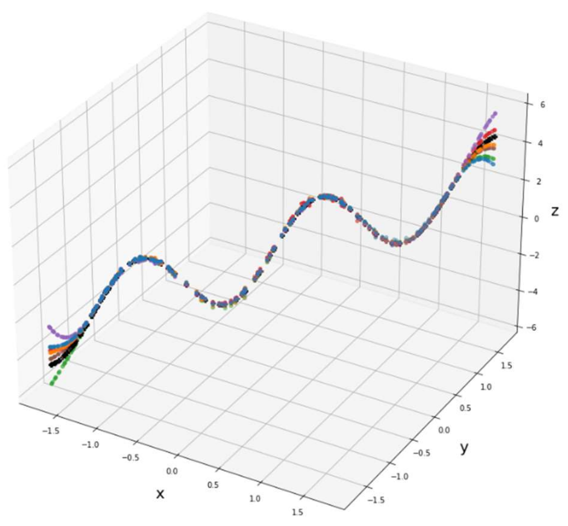


Figure 6 (left). *Samples of predictive functions taken from the prior distribution (shown in colours), with a mean of 1 and covariance determined by the test set. The actual function is shown in black.* ***Figure 7 (right).*** *Samples of functions taken form the posterior distribution (coloured lines). Given the training data, the predictive functions are much more accurate, with all perfectly following the actual function within the range of observed training data.*

To constrain the prior distribution, functions are limited to those with a specific mean (usually zero) and certain level of smoothness - the measure of similarity between outputs corresponding to nearby inputs. This similarity is determined by the GP kernel, which calculates the **Σ** elements. The choice of kernel allows prior knowledge to be incorporated into the distribution – e.g. whether the function is periodic, linear or noisy.

Then, given some additional training data with known outputs, the distribution is updated to a more accurate posterior distribution (PD) (*Figure 7*). This PD is defined by its new $\boldsymbol{\mu}^*$ and $\boldsymbol{\Sigma}^*$, and is calculated by conditioning the prior distribution on the known training data (Bayesian inference). This effectively confines the set of possible functions to those that pass through the training points. The model's predictions are then provided by the PD, with each being a 1D Gaussian distribution of over possible outputs. However, the mean predictions are usually taken from $\boldsymbol{\mu}^*$, with $\boldsymbol{\Sigma}^*$ acting as a confidence indicator (its diagonal terms represent the variance of each mean prediction).

5. Which additional algorithm did you choose and why?

Since the input data appears to have quite a complex relationship to crop yield and a cluster-like structure (*Figure 4*), where nearby observations tend to give similar crop yields, I decided to use the K-Nearest Neighbours algorithm (KNN) with a Euclidian distance metric. Due to its non-parametric nature, KNN makes no assumptions about the data's structure (only that distances can be calculated) and hence should work well with the dataset's complex, clustered structure. Additionally, being a "lazy" algorithm, KNN has no explicit training phase, meaning that any test predictions consider the most recent training data and hyperparameters. This simplifies hyperparameter optimisation and, combined with the fact that KNN generally only takes one hyperparameter, helps to reduce tuning time – an important factor given the large number of observations.

Furthermore, KNN with Euclidian distance is designed for problems, such as this one, with continuous variables. The dataset's reduced number of features (16) after PCA is also congruent with KNN, which works well with low-dimensional problems but begins to struggle with time/memory consumption and the curse of dimensionality if $n_{featrues} \gtrsim 100$.

6. What are the pros and cons of the algorithms?

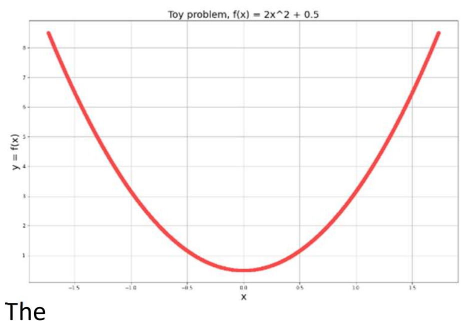
The main benefit of LR is its fast convergence, with gradient decent it has the shortest runtime of all four algorithms. However, my LR algorithm is limited to predicting linear relationships, which is a major drawback because it cannot pick up on more complex relationships, and therefore strongly underfits non-linear problems.

Compared to LR, RFs can highlight a much wider, more complex, variety of trends. Unlike the other algorithms, RFs are also able to ignore unnecessary features, making them better candidates for problems with high proportions of useless features. However, depending on the number of trees, RFs can be slower than both LR and KNNs, and are also more prone to under and overfitting than KNN and GPs, making hyperparameter tuning crucial. RFs and LR both also benefit from explainable models, where the reasoning for predictions can be more easily visualised and hence understood than GPs and KNN.

On the other hand, GP is the only algorithm able to quantify the uncertainty in its predictions, and the only one that allows prior knowledge of the relationships to be built into the model. However, it is by far the most computationally expensive algorithm, making training on larger datasets unfeasible. GP can also be strongly biased when the dataset contains several outliers, hence pre-processing is essential.

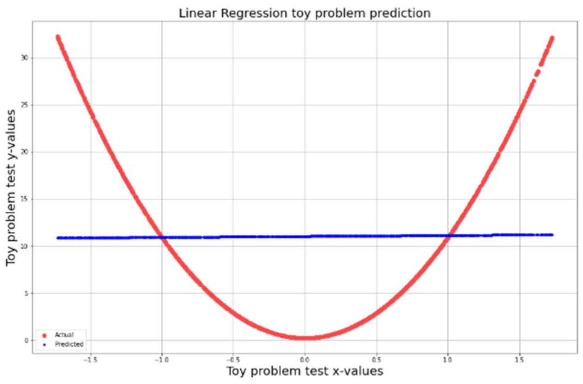
Finally, KNN and GP are "lazy" learners with no explicit training phase, so must store and iterate through all training observations whenever predictions are required. Unlike RFs and LR, this allows them to predict using the most recent data available, however, prediction times are therefore far slower than RFs and LR.

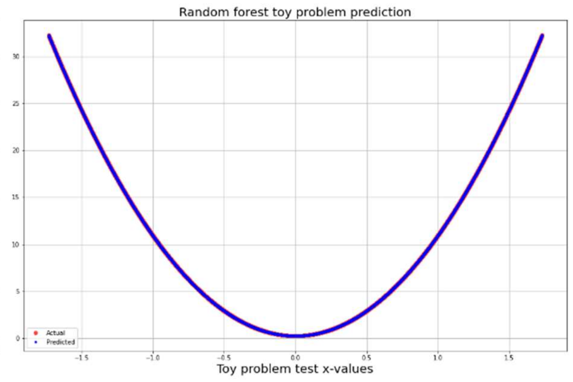
7. Describe the toy problem used to validate the algorithms, and explain its design?

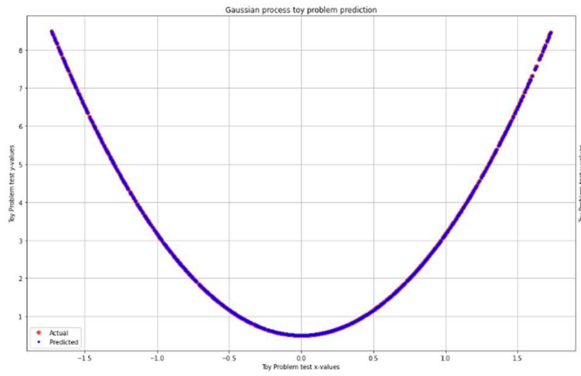


I decided to use the function $y = 2x^2 + 0.5$ for $x \in [-2, 2]$ as my toy problem (as shown in *Figure 8*). The reasons for this are that, having only 2 dimensions, the function is easy to visualise, allowing for a direct visual comparison between actual and predicted outputs to help identify and debug issues. Being a simple quadratic, the function's correct solutions are easily calculated, making such comparisons straight forward and allowing for measurements such as RMSE to be confidently determined. The function's quadratic and symmetric nature also means that LR is expected to fail to fit it and output a flat line with a gradient of zero. The other algorithms should, however, fit it almost perfectly. I also decided to offset the function by 0.5 in the y-direction, as this would allow any systematic error in predictions to be easily visualised.

8. What evidence of correct, or incorrect, implementation did the toy problem provide?







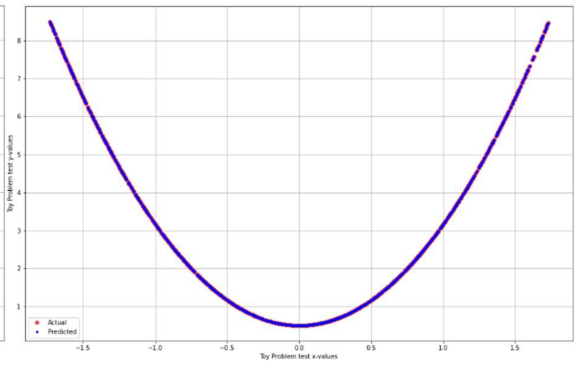


Figure 9 (top left). *This shows the linear regression prediction for the toy problem described above. This is as expected since my LR can only predict functions of the form* $\boldsymbol{y} = \boldsymbol{wX} + c$ *(where* $\boldsymbol{w}$ *is a constant vector and* c *is a constant scalar).* ***Figure 10 (top right).*** *This shows the random forest toy problem prediction, after correcting the issue shown in Figure 13. It is as expected, with the RMSE tending towards 0 as the maximum tree depth increases.* ***Figure 11 (bottom left).*** *This shows the Gaussian process toy problem prediction. It is as expected, with the posterior mean function producing an RMSE of ≈ 0.* ***Figure 12 (bottom right).*** *This shows the KNN toy problem prediction. It is also as expected, with close inputs corresponding to close outputs, and an RMSE of ≈ 0.*

Since regular linear regression (LR) can only predict linear functions, and the toy problem is symmetrical about the y-axis, I would expect its prediction to be a straight line with a gradient of zero. As shown in *Figure 9*, this is exactly what the algorithm produced, indicating correct implementation.

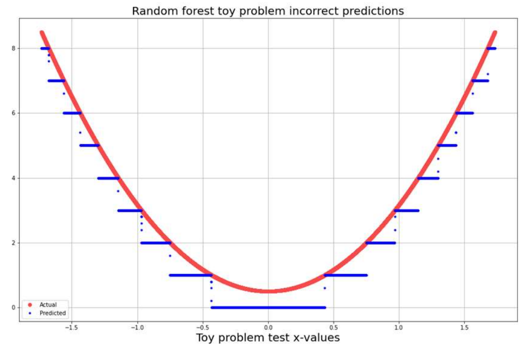


Figure 13. *This shows the initial predictions produced by my random forest before troubleshooting. All predictions were at integer values, it turned out this was duie to my function wrongly rounding each mean precision to the nearest integer. After seeing these toy problem results, the issue was resolved and the RF gave the result shown in Figure 10.*

Using a RF, I would expect the algorithm to divide the input range into n segments and predict the mean output of the training data for each. For minimum split and leaf sizes of 1, as maximum depth (d) increases I would expect n to increase, and the predictions to move closer towards the actual function. However, initially the RF did not behave as expected, with n not increasing beyond 17 (see *Figure 13*). Using the toy problem, I realised the RF was wrongly rounding mean predictions to their nearest integers. Once corrected, the RF behaved as expected (see *Figure 11*).

The GP toy problem predictions behaved as expected (*Figure 11*), with the prior distribution producing a range of possible functions (see *Figure 6*) and the posterior mean function producing an RMSE of ≈ 0.

KNN predictions were also as expected (*Figure 12*), with the close toy function inputs being correctly correlated and mapped to close output predictions, resulting in an RMSE of ≈ 0.

9. How were the hyperparameters optimised?

With all algorithms, an initial random search of hyperparameters was performed in order to efficiently identify the best hyperparameter regions. This was followed by individual linear searches of hyperparameters, with all but one set to the optimal values previously determined by random search. Hyperparameter plots were then generated from these linear searched to identify their relationships to the RMSE. All optimisation was performed on the validation set to avoid implicit overfitting.

Linear Regression

Since gradient decent was used, learning-rate can be optimised to find the most time-efficient value that is not too large it overshoots the optimal solution. A linear log-space search of learning-rates was conducted. *Figure 14* compares their convergence curves.

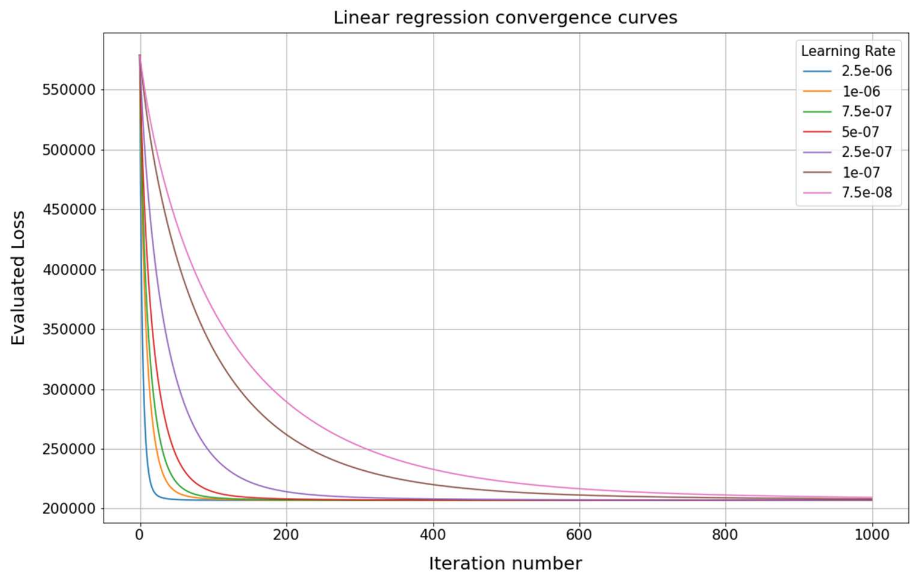
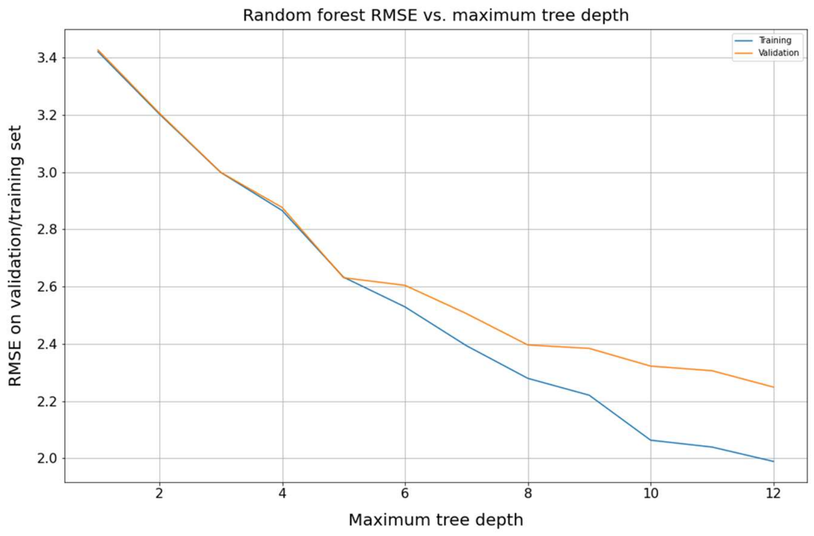
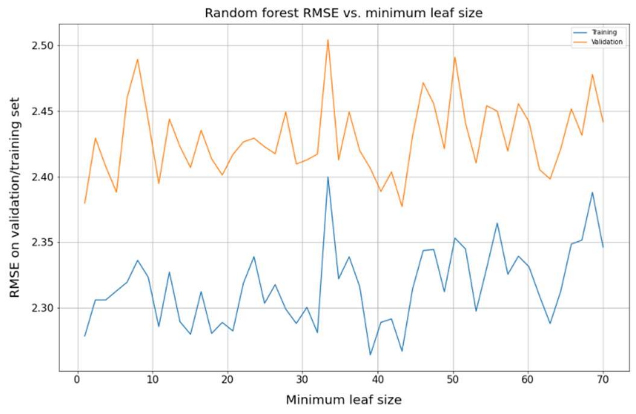


Figure 14. *This shows the rate of convergence of gradient decent to the (local) optimal loss value, using various learning rates. Each learning-rate is shown in a different colour. The fastest converging value proved to be a learning-rate of 2.5x10^{-6}, with any larger values causing the loss to diverge to infinity.*

Random Forest

Here I tuned the max-tree-depth (Figure 15), min-leaf-size (Figure 16) and min-split-size (Figure 17). The optimal tree depth is not purely about the validation set's RMSE, over and under fitting should also be considered. *Figure 15* clearly shows that when $d \geq 6$ then model begins to overfit (training RMSE > validation RMSE), and below that the model is underfitting. The optimal trade-off appears to lie at $d = 8$.





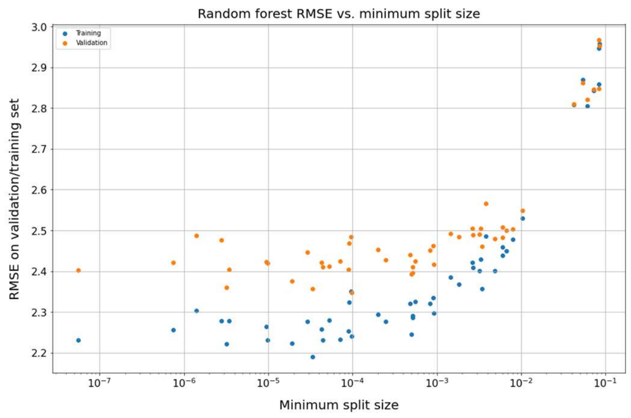
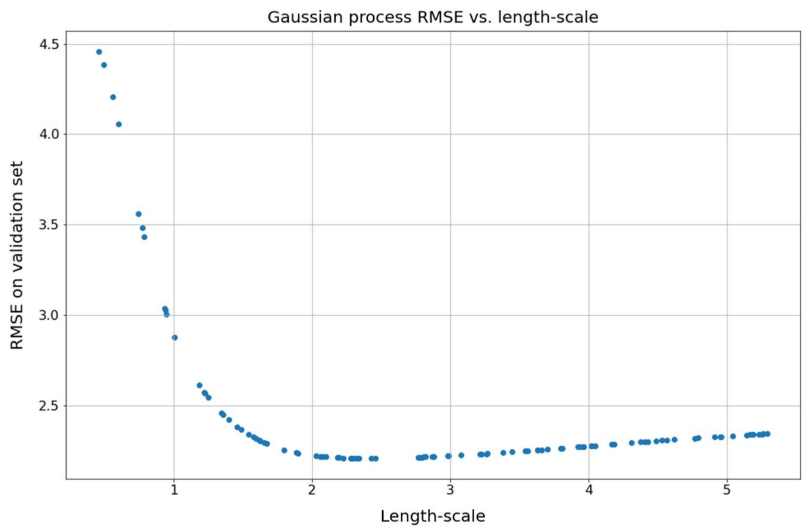
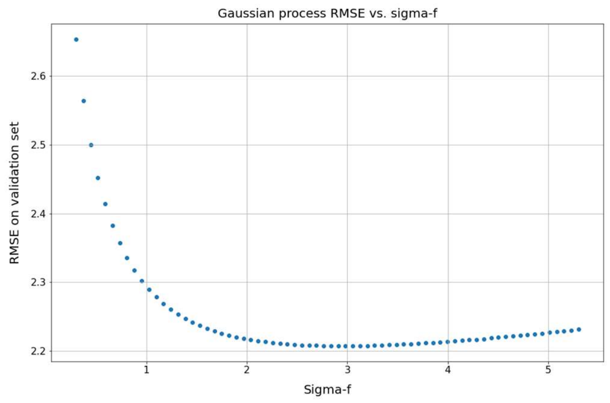


Figure 15 (top). *A graph of RMSE vs. maximum tree depth for the random forest (training RMSE in blue and validation RMSE in orange). As maximum tree depth increases, the resulting RMSE of predictions decreases. However, at a depth of 6 the training RMSE starts to decrease at a greater rate than that of the validation set. This meaning that the forest begins to overfit on training data. The optimal trade-off between under and overfitting is at a depth of 8.* ***Figure 16 (bottom left).*** *This shows the relationship between minimum left size and RMSE. Optimisation showed that this hyperparameter's effect varies with the random nature of random forest bagging, although lower minimum leaf sizes tended to perform better.* ***Figure 17 (bottom right).*** *This shows the relationship between minimum split size and RMSE. There is a optimal split sizes were found to be around 10^{-4}.*

Gaussian Process

Highly dependent on length-scale hyperparameter. *Figures 18,19,20* show linear searches of each hyperparameter after the initial random search, using 5000 observation representative sample.





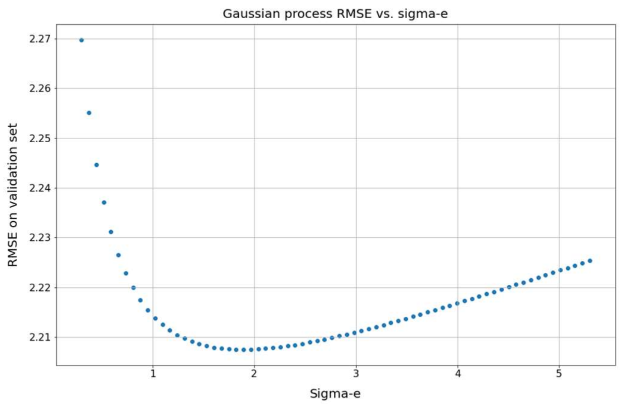


Figure 18 (top). *A graph of RMSE vs. length-scale for the Gaussian process. Optimal length-scale was found to be at 2.4.*
Figure 19 (bottom left). *A graph of RMSE vs. sigma-f for the Gaussian process Optimal sigma-f was found to be at 2.98.*
Figure 20 (bottom right). *A graph of RMSE vs. sigma-e for the Gaussian process Optimal sigma-e was found to be at 1.85.*

KNN

Best RMSE with k=12 and inverse-distance weighting, with inverse-distance weighting performing better all-round

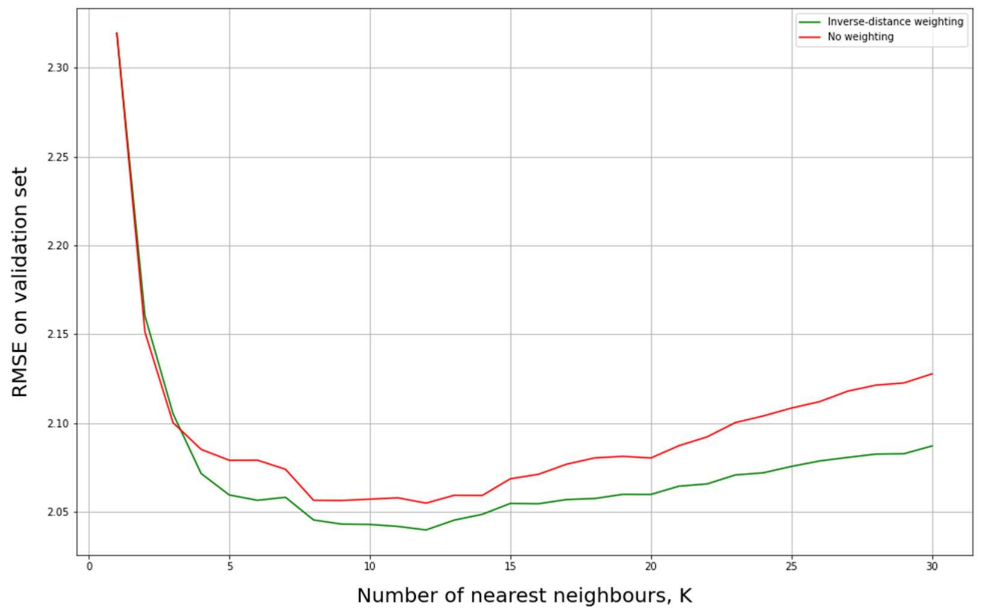


Figure 18 (top). *A graph of RMSE vs. the number of nearest-neighbours, k, for both no distance weighting (red) and inverse-distance weighting (green). Optimisation showed that the optimal k is 12, and that inverse weighting improves RMSE for all k values.*

10. What results are obtained by the algorithms?

To determine the accuracy of results obtained and compare this between algorithms, I decided to calculate the RMSE, MAE and R^2 for each. These statistics are shown in (*Figure 22*). Since RMSE is more strongly influenced by larger errors than MAE, it can help to show differences in the scale of errors generated.

	Linear Regression	Random Forest	Gaussian Process (5000 training observations)	KNN
RMSE	3.003	2.309	2.207	2.039
MAE	2.121	1.563	1.467	1.235
R^2	0.307	0.591	0.627	0.681

Figure 22. *Table showing the best obtained prediction statistics after hyperparameter optimisation. LR is using a learning rate of 1x10^{-6}; RF is using 50 trees, max depth = 8, min leaf size = 7, min split fraction = 6x10^{-4}; GP is using length-scale = 2.4, sigma-f = 2.98, sigma-e = 1.85; KNN is using k=12 with inverse-distance weighting.*

KNN has by far the best RMSE, MAE and R^2, meaning that its results are the most accurate both in terms of average error but also the reduction of large errors (as shown by the reduced RMSE and in the absolute error histogram of *Figure 19*). Linear regression was the worst performing algorithm in terms of all three statistics. This is likely due to the non-linear nature of the problem. The GP and RF produced a similar accuracy of predictions, with GP performing slightly better despite using only 5000 training observations.

11. How fast do the algorithms run and how fast could they run?

Linear Regression

To train, gradient decent depends on the number of optimisation iterations, costing $O(k_{iters}N^2)$, where N is the number of training observations. Predictions then cost $O(N_{test})$. Using gradient decent with an optimiser learning rate, my LR's mean training and prediction runtimes are 2.9 and 0.0 seconds.

Random Forest

Training complexity is $O(Nfdt)$, where f = number of features sub-sampled, d = maximum tree depth and t = number of trees. Prediction complexity is $O(N_{test}dt)$. With 50 trees and maximum depth of 8, my RF's mean training and prediction runtimes are 26.5 minutes and 0.18 seconds. This runtime could be improved by parallelising trees (reducing runtime by a factor of $n_{parallel\ trees}$) and vectorising the RF functions to make use of optimised NumPy code.

Gaussian Process

Overall complexity is $O(N^3)$ due to its matrix inversion, with mean function predictions then taking $O(N_{test})$. Initially, my average runtime was 290 seconds with N = 5000, however, after vectorising the kernel this was reduced to just 4 seconds – over 70 times faster. Furthermore, I replaced matrix inversion with a Cholesky factor, reducing the overall complexity to $O(\frac{2}{3}N^3)$.

K-Nearest Neighbours

Complexity is $O(kNf)$), where f = the number of features and k = number of nearest neighbours. With k = 13, my KNN's mean runtime is 19.6 seconds.

12. Which algorithm would you deploy and why?

Out of the four algorithms, I would deploy KNN. This is due to KNN providing both the lowest RMSE and MAE and the highest R^2 score, making it by far the most accurate algorithm (7.6% lower RMSE than GP, the next best). Given its lower RMSE, and tighter absolute error (AE) distribution (*Figure 19*), KNN resulted in reduced extremes of AE, making it a more reliable predictor of crop yield. Alongside this, with an average runtime of 19.6 seconds, KNN was also the 2nd fastest algorithm to run on the full dataset. When combined with the fact that KNN is a "lazy" learner, I believe it to be the most flexible algorithm in terms of updating predictions quickly and accurately if new training observations were to be added, making it the most "future-proof" of all the algorithms.

13. How could the best algorithm be improved further?

To further improve prediction accuracy of KNN, the inverse-distance weighting currently used ($w = \frac{1}{d+\varepsilon}$, where ε is a small constant) could be updated to one in which the variable $d_{1/2}$ is used (see *Eqn. 1*).

$$w = \frac{d_{1/2}}{d_{1/2} + d} \qquad (\textit{Eqn. 1})$$

This updated weight formula means that when the distance = $d_{1/2}$, the weight falls to $\frac{1}{2}$, and the maximum weight is 1. The variable $d_{1/2}$ could then be optimised as a hyperparameter, allowing the scale of the weighting to be tuned to the dataset, which would likely improve overall accuracy.

As well as this, n-fold cross validation could be employed to more accurately tune the hyperparameters (k and $d_{1/2}$) and reduce overfitting, resulting in a more accurate and generalised model. Overfitting would be reduced because the hyperparameters would be averaged over n validation sets across all observations, instead of just a single subset of observations.

14. If you were to try another algorithm then which one and why?

I would also try a Support Vector Machine (SVM). The main reason for this is that the dataset's features have a complex, non-linear relationship with crop yield (*Figure 4*). An SVM should be able to utilise the kernel trick to project the data into higher dimensional spaces, allowing it to more-easily identify non-linear trends using hyperplanes. Moreover, SVMs work well with high-dimensional data, so would be ideal for this 37 feature dataset.

15. Are the results good enough for real world use?

Here I assume that yield predictions are used to inform an agricultural risk management system, and that people's lives and livestock depend on having enough of the crops in question. If this is the case, it is crucial that the results are as accurate as possible since incorrect predictions could be life-threating. For instance, if crop yields are over-estimated, the risk of shortages could be deemed low and backup supplies could be cut-back, resulting in a potential famine when the actual yields turn out to be far lower. My best algorithm's RMSE and MAE are 2.039 and 1.235, which is $\approx$56% and $\approx$34% of the mean crop yield respectively. Being more influenced by larger errors, the higher RMSE reflects the skewed nature of the error distribution – meaning that when the algorithm goes wrong, it does so with large errors. This is certainly not suitable for critical crop yield predictions, where a more consistent error distribution is required to avoid the scenarios above.

16. How could this solution fail?

In order to use SVM, a kernel must be selected based on known assumptions about the model. If these underlying assumptions are incorrect, the kernel chosen may not be the best for the specific problem, potentially leading to large prediction errors. Additionally, SVM works by finding distances to the hyperplane, so is sensitive to the scale of features. SVM could fail if the features have different scales and feature scaling was not applied.

17. What improvements could be made to the data set?

Additional features could be added, however, these should be sufficiently independent of the original variables, as otherwise they would not provide any further information. Such features could include the pest population size, prevalence of crop diseases and air pollution; not only would crop yield likely be inversely correlated with all three, but they should also be largely independent of the original features – maximising the additional information they carry. Being directly linked to the size of crop yields, soil fertility and pH would also be insightful additional features.

Alongside the numerical features, satellite imagery of crop regions and their topography could be added. Algorithms such as ANNs could use this to find trends and combine predictions those of numerical data.

Aside from new features, the dataset could also be updated to include measurement uncertainties, which could be used to inform the extent of regularisation required.