An Investigation of Current Machine Learning Techniques for Predicting Plant Phenotypes from Genotypes

CS39440 Major Project Report

Author: Felix Andrade May (fea6@aber.ac.uk)
Supervisor: Prof. Reyer Zwiggelaar (rrz@aber.ac.uk)

2024–08–08
Version: 1.1.0 (Release)

This report was submitted as partial fulfilment of a BSc degree in Computer Science and Artificial Intelligence (with Integrated Year in Industry) (GG47)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, U.K.

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Felix Andrade May

2024–08–03

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Felix Andrade May

2024–08–03

Acknowledgements

Melissa

Socks

Reyer Zwiggelaar, for his patience and encouragement

James Strong and Stephen Gow, for their help and knowledge

Abstract

An impending climate crisis and a growing global population are contributing to a growing demand for energy. Without consideration from governments and industries, non-renewable resources will continue to be used to fuel economies and to mitigate the physical effects of shifting climates on populations. Renewable sources of energy are already being adopted at increasing rates [1,2], however the long-term maintenance of these methods is often under-evaluated before implementation. A potential solution to avoid rising costs of hardware replacement and maintenance is to expand the use of biofuels, which can be grown on existing farmland across the globe. Miscanthus sinensis, a perennial grass, is well suited to growing in a wide range of climates. Its high rate of growth in tight clusters makes Miscanthus sinensis a good candidate for renewable bioenergy production [3].

Machine Learning (ML) techniques can be used to predict the presence and values of phenotypes, which are the physical aspects of the plant, the from plant genotypes [4–7]. Without the use of ML models, phenotypes can be hard to predict because of the vast range of information that can effect them. This information falls into two broad categories: genomic information, such as the appearance and composition of Single Nucleotide Polymorphisms (SNPs) in the genome; and the environmental information, which incorporates all the non-genomic information such as the amount of available nutrients in soil or environmental stresses such as strong winds and storms. By applying ML methods to a dataset of Miscanthus sinensis genotypes and phenotypes a better understanding of the relationship between the genetic composition of the plant and the phenotypes can be gained. This improved knowledge would allow for selective breeding and hybridisation of Miscanthus sinensis by industrial growers to improve the growth time, yield, and environmental resistance of the plant, subsequently increasing the potential amount of renewable biofuels that can be produced from the plant.

This paper describes the development and evaluation of eight ML models, as well as Command Line Interface (CLI) programs to run the models and the development process used in their creation. The models were trained on a dataset of 110 Miscanthus sinensis genomes,containing 13,630 Single Nucleotide Polymorphisms each, to identify the measured values of thirteen Miscanthus sinensis phenotypes. Ridge Regression (RR) ($\mu R^2 = 0.26$; $\mu Recall = 0.31$) and Support Vector Machines (SVMs) ($\mu R^2 = 0.24 - 0.34$; $\mu Recall = 0.28 - 0.30$) were most effective at predicting phenotypic values compared to Artificial Neural Networks (ANNs) and other linear models, with high recall rates (> 0.70) achieved by the SVM models in some scenarios.

Contents

Chapter 1

Introduction

As the global population grows so does the demand for the production of energy. Combined with a shifting climate, the need to improve sources of renewable energy becomes ever more important. Miscanthus sinensis is a perennial flowering grass native to east Asia. It's ability to grow quickly and in dense clusters, even in high-stress environments, makes it a good candidate for renewable bioenergy production [3].

Machine Learning (ML) is the application of algorithms to optimise a solution from a given input and from experience, which is previous information given to the algorithm. This information is typically in the form of pre-labelled information, like information plaques in a museum. These algorithms are defined by sets of parameters which can be tuned to improve the ability of the algorithm to optimise the solution. An algorithm that has been trained on the input data becomes a model which can be given new, unseen, data from which it will attempt to predict an outcome. This form of learning is called supervised learning, which differs from the semi-supervised paradigm, where only some of the training data is labelled, and unsupervised learning paradigm, where none of the training data is labelled [8].

Better understanding the link between the genotypes of plants and their phenotypes will improve our ability to selectively breed crops to improve yields, decrease growing times, and improve disease resistance. Machine Learning (ML) techniques have proved useful for genotype to phenotype prediction in plants [4–7]. Phenotypes, the physical aspects of an organism, can be hard to predict without the use of ML due to the wide ranging amount of information which contributes to their manifestation. This information falls into two categories, genomic information and environmental information. Genomic information is any information contained within the organism's biology, such as the appearance and content of Single Nucleotide Polymorphisms (SNPs) in the genome. This paper looks only at the influence and relation of genomic information to phenotypes. The second category,environmental information, is any non-genomic information that could effect the manifestation of phenotypes, such as the available nutrients in growing stages, or the amount of available sunlight. It also encompasses environmental stresses such as, in the case of plants, strong winds and storms which may damage the plant or hinder its development, thus altering its phenotypes. If a model that accurately estimates phenotype values from the genotype can be developed, then it can be employed both in the renewable biofuel fuel industry

as well as in further research. As the cost of commercially sequencing whole plant genotypes continues to decrease towards and past the $1,000 (USD) mark [9], a generally effective model for phenotype from genotype predictions would allow sufficiently motivated industrial growers to better understand why and how their crops grow. By sequencing their bioenergy crop and training a model on the physical properties of the same sampled crops, industrial growers would be able to combine the traits of the different plants, through either genetic modification or traditional selective breeding. As an example, an industrial grower who has two fields of the same crop may find they have a field that grows tall and fast and a field that grows slow and stout. The industrial grower may want to identify the genotypes that contribute to the manifestation of these phenotypes. If a model exists that can accurately predict quantitative phenotypic values from the genotype, then the industrial grower will be able to splice the plants of these two fields together to produce a plant that grows quickly into a tall and sturdy crop. In the case of bioenergy, this would allow the grower to increase the yield of their crops and subsequently sell them for a greater commercial profit from the same acreage as before. This increased yield would also increase the potential amount of renewable biofuels that could be produced and utilised in the energy production and transport sectors, helping companies, local governments, and national governments actually achieve their every approaching climate targets.

This paper uses a genome wide study of Miscanthus sinensis combined with physical measurements of thirteen phenotypes. Eight Machine Learning regression techniques, including a Ridge Regression (RR) model which was used as a baseline comparator, were developed and trained to predict the quantitative values of the phenotypes from the genomic information. The ability of the models was assessed by evaluating the models' R^2, recall rate, and confusion matrices. The average performance of the models in those metrics can be used to infer a correlation between the genomic composition of the plants and their physical attributes. This can be done because a trait which models consistently perform poorly in can be said to have little correlation to the genomic aspect of the plant and instead may be far more correlated to the environmental aspects the plant existed within, though this paper does not take the environmental aspects into consideration. A Ridge Regression (RR) model was used to set a baseline ability, as the linear Ridge Regression model is equivalent to a BLUP model [10, 11], one of the more popular models employed in the field of bioinformatic prediction.

A range of agile techniques [12] were used to manage the project. KanBan boards with WiP limits of five were created to both organise and limit the active workload. Epics outlined in a project overview were used to create more specific and detailed cases that were worked on during informal sprints. Version control services BitBucket, Git, and Gitk were used in combination with the project management service Jira to manage, host, and maintain the project [13–15].

Chapter 2

Methods and Methodologies

2.1 The Data

Models were trained on a genetic wide dataset of Miscanthus sinensis grass, composed of two linked datasets. The data originates from a Near Infra-red Spectrum (NIRS) study of the grass [16,17]. The first dataset contains Phenotype information, with records for thirteen phenotypes: Average Senesence (AvgeSen); Base Diameter (BaseDiameter); Canopy Height on day 119, 158, and 176 of the year 2008 CanHght.119, CanHght.158, CanHght.176; Day of the year that Flowering Stage One was observed (DOYFS1); Dry Matter (DryMatter); Day of the year that Emergence Stage Four was observed (es.4.doy); Maximum Observed Canopy Height (MaxCanopyHeight); Moisture (Moisture); Stem Diameter (StemDiameter); Tallest Stem (TallestStem); Transect Count (TransectCount). This phenotype information is supported by some additional auxiliary information about the Replicate Block, Field Order, Plant Block Row Position, Plant Block Column Position, Species, Genotypic Plate ID Number, Order, Hierarchical Population Genetic Structure, Growing Longitude, Growing Latitude, and Growing Altitude. There is also a record of the SNP Genotype, represented using an identifier such as `Mb-000`. A similar series exists in the second part of the Miscanthus data. This makes it possible for the two files to be linked and aligned, so that the first row of SNP information is corresponds to the first row of phenotypes. The Phenotype dataset consist of 552 entries each with values for the twelve auxiliary information and thirteen Phenotypes listed above as well as a column for the SNP Genotype codes. Each of the 138 SNP Genotypes is represented four times in this dataset because the samples were cut into four segments during collection. It should be noted that some Genotypes present in the Phenotype dataset are not represented in the SNP Genotype dataset. A description of each of the phenotypes can be found in Table C.1, Appendix C.

The second part of the Miscanthus sinensis data is a dataset of of 861 SNP Genotypes. Each SNP is encoded using the presence of 13,630 non-reference (minor) alleles which are given a value of 0, 1, or 2. That is: from a reference SNP allele, if there is one mutation, the column will be marked as 1, as demonstrated in Table 2.1. While this data originally contained 861 entries, this was reduced to 110 entries using a short Python script [Listing B.5] to filter out any SNP Genotypes that did not appear in the Phenotype

Actual SNP	Numeric Encoding
AA*	0
AT	1
CT	2
CC	2
CA	1
AA	0

Table 2.1: How SNPs are numerically encoded. In this example, the first SNP (AA), is the reference SNP and the following SNPs are measured against it.

dataset. Due to the construction of these two datasets, no further feature selection was performed on the SNP Genotypes dataset before being passed to the models.

2.2 The Environment

Models were run in a containerised environment using Docker [18]. A script (`start_container.sh` [Listing B.2]) is used to build and start the environment container. The script begins by building a Docker Image from the Dockerfile [Listing B.1]. The Dockerfile contains the build instructions to create a Docker Image running the Ubuntu Noble 24.04 operating system, with several packages installed, including: Python3.12, pip, vim, wget, and their dependencies. The Dockerfile also uses pip to install the Python packages necessary for the models to run, including SK-Learn, Keras, and Pandas [19–22]. A single script was chosen for this process as it greatly simplifies the start-up experience for the user, requiring them to only run one command in the CLI compared to the five long and repetitive commands required to complete the process. From here on, this testing and development container will be referred to as 'the container'.

Docker was chosen as it is a lightweight and easy to install technology; requiring only an installation of the Docker Engine, which can be done by downloading the Docker Desktop client [23]. By containerising the development and testing environment we are able to run each model in as identical an environment as possible each time. It also allows for others to easily run the models, as every third party library or package is installed into the container when it is built, rather than requiring them to be manually installed by the user on their own machine.

Each time a new package or library was needed for development, a new container would be built. The new package would then be manually installed into this container from the attached Command Line Interface (CLI) and verified. Once the installation process was confirmed, the same steps could be added into the Dockerfile itself. After this, the container would be built again and the same manual verification would take place to ensure the correct third party packages and dependencies had been installed. While conventional unit testing of a Dockerfile is possible [24–26], I decided that the simplicity of the requirements for the Dockerfile as well as the static nature of the file outweighed the complexity of these third party unit testing solutions. When the changes had been made, the `version` variable of `start_container.sh` would be incremented

by one to mark the update.

2.3 The Models

Once the container had been initialised and entered through the CLI, models can be run using PROject moDELS (PRODELS) [B.4]. This is a simple script that accepts several predefined flags to define the input arguments. By using flags it makes entering the correct arguments easier for the user, because they can be entered in any order, and then PRODELS will pass the arguments to the model in the correct order. If the models were instead run directly by calling Python, then each of the arguments would have to be entered into the CLI in the same order the program expects every time it needed to be run. PRODELS also provides a 'help' flag, which displays a formatted message to assist the user in remembering the program's flags and the arguments for each flag, as seen in Listing 2.1.

```
PRODELS - PROject moDELS 0.7.1

Usage: prodels [options] [file ..]  runs a given model on given datasets

Arguments:
    -d {0|1},             0 by default runs the model in debug mode if set to 1
    -h,                   displays this helpful message :)
    -m (with file name),  the name of the Model to train
    -o (with file name),  the name of the directory with /environment/outputs/
   to write results to, does not need to exist
    -p (with file name),  the full path of the Phenotype CSV dataset
    -s (with file name),  the full path of the SNP CSV dataset
    -v,                   displays the program version
```

Listing 2.1: The message displayed after running `./prodels -h`

The models were developed using Python, which was chosen as it is a lightweight, but powerful, programming language that specialises in scientific programming. The dynamically typed syntax and focus on whitespace makes the language easier to read for non-software-developers in a print format such as this. Python also includes a large standard library and makes it easy to install more packages to the system by using the pip package manager [27]. Compared to other languages that are popular in the field of Phenotype prediction, namely R [28], Python is more versatile and has a simpler installation process across a wide range of Operating Systems compared to R. Although the process can be somewhat automated within R scripts themselves, the installation of third-party libraries to Python is also simpler and more efficient compared to R. This makes it a sensible choice to use within the container.

Two utility files were created to aid in the function of the models. One file, `process.py`, contains a selection of methods responsible for loading and configuring the data, as well as creating plots and graphs such as confusion matrices and diagrams of the decision tree and Artificial Neural Network (ANN) models. This file is tested with the PyTest library [29], which is an open-source testing framework for Python. It provides a detailed and easy to read output compared to the output produced by Python and the unittest package. The tests are run through another script, `run_tests`,

which runs the test files themselves with the `-v` option to return a more detailed, verbose, log of test results back to the CLI. `process.py` also creates confusion matrices for each phenotype, these are saved to a directory called `cmats`, which is found in the output directory given to PRODELS by the user. The second file, `write_files.py` is responsible for writing any new files created as a result of running a model. It will automatically create any needed directories that do not yet exist within the container before writing files to them. In order to save these output files to the host machine the user must use Docker to copy the files out, which can be done using the `docker cp` command [Listing B.3]

The two datasets were processed using the Pandas library [21]. Pandas is an open-source data analysis package for Python and was chosen primarily for its DataFrame class [30]. DataFrames are fast and efficient objects that form a key feature of Pandas, they allow for much more efficient data processing and manipulation than Python's own dictionaries and multi-dimensional lists. The datasets are read into the models using Panda's `read_csv` function. This reads a Comma-separated Value (CSV) file into the program as a DataFrame. Two other functions in `process.py` are then called upon to perform additional pre-processing of the data, these align the DataFrames by the '`geno`' column and replace all N/A values with 0s. These pre-processing functions also rely on calls to Pandas functions designed for such implementations.

Two primary libraries were used in the creation of the machine learning models. The Ridge Regression (RR), Decision Tree, Least Absolute Shrinkage and Selection Operator (Lasso), and Support Vector Regression (SVR) Models utilise the Scikit Learn library [19] for the model architecture itself. The three Artificial Neural Networks (ANNs) are built using Keras [20]. Each model was tuned using Scikit Learn's GridSearchCV function [31]. A five fold cross-validation was performed with a range of possible parameter values specific to each model. Once this was complete and the results of the best model were written to the output file for each phenotype, the median value for each parameter would be used to train and fit the model again for a final time, the results of these models are shown in Chapter 3. If there was no clear median value, a mean value would calculated and implemented instead. A median value was chosen for the final model as each model was trained and evaluated against each Phenotype separately. Because of this, the most common results of all cross-validation parameter selections should be consolidated into a single model that will be trained and evaluated on each Phenotype again for the final results, as this would be the most representative set of parameters for the model. Some parameters that are common to all or most of the Scikit Learn models, such as 'tolerance', were eliminated as tuneable parameters early on in the model refinement process for having little impact on model performance and were excluded from further development. Many of Scikit Learn's models and utility functions also provide a 'random_state' parameter; in all cases where this was a valid parameter the value was set to 611, to ensure the model's architecture and results are reproducible.

The performance of the models was evaluated using two primary metrics, the R^2 and the recall rate. The R^2 was also calculated using Scikit Learn [32]. Because the R^2 measures the ability of linear regressors, some of which do not include a y intercept, the best possible R^2 would be a value of 1.0, however the R^2 can be infinitely negative because the model can be arbitrarily worse. The recall rate is calculated as the ratio between the number of true positive predictions to the sum of true positives and false

negatives predictions. The rates of true positives and false negatives are calculated from the confusion matrix created by the `process.py`. The recall rate is strictly positive, so the best possible value is 1 while the worst possible score is only 0. The confusion matrices are also used to assess the performance, and can be found in Appendix D.

Scikit Learn is an open-source machine learning library for Python that offers a wide range of linear and non-linear models, scoring functions, and utility functions. The models are highly tuneable, particularly when used in combination with the library's GridSearchCV [31] function, which allows for arrays of parameters to be exhaustively searched through. Once the grid search is complete a single model is returned using the combination of parameters that produced the best model on the data using the specified scoring metric. Scikit Learn also provides a large selection of scoring metrics for models [33]. Because these metrics only require an array of true values and predicted values they can be used to score both Scikit Learn based models, as well as models built from other libraries, such as Keras.

Keras is a lightweight machine learning library for Python specialising in neural network architectures. Keras provides a frontend that can be configured to work with a range of popular machine learning frameworks such as TensorFlow, JAX, and PyTorch. The accessible API of Keras also makes it possible for custom nodes to be used in an ANN's layers. This was important when it came to developing the Radial Basis Function (RBF) Neural Network model.

2.3.1 Model Specifications

Ridge Regression Model

The Best Linear Unbiased Predictor (BLUP) model is a long standing method for genomic prediction. Made popular in the field of genomic prediction by Meuwissen et al. [10], BLUP relies on Genetic Breeding Values (GBV) to define the covariance between known relatives. When these variances are made to be equal to each other, and a uniform prior is applied to the prior distribution, BLUP is equivalent to a Ridge Regression (RR) model [11]. The Ridge Regression model, implemented with Scikit Learn, uses a linear least squares algorithm with L2 regularisation to minimise the objective function seen in Equation 1 [34].

$$||y - Xw||_2^2 + alpha * ||w||_2^2 \tag{1}$$

Another key model in this field is the BayesB model [10]. This form of Bayesian prediction makes use of statistical sampling methods, often Gibbs sampling, to inform the Bayesian priors. Previous work has shown Ridge Regression to be as effective in Genomic prediction as BayesB [35–39]. As a result, this Ridge Regression model will be used as a baseline comparator in Chapter 3 because it is an appropriate approximation of the ability and function of the two most popular ML models in the field. The full parameters of the model can be seen in Table 2.2.

Parameter	Cross-Validation Values	Final Configuration
Alpha	0.001, 0.01, 0.1, 1, 10, 100, 1000	1000
Copy X	True	True
Fit Intercept	False, True	False
Max Iterations	None	None
Positive	False	False
Random State	611	611
Solver	svd, cholesky, lsqr, sparse_cg, sag, saga, lbfgs	saga
Tolerance	0.0001	0.0001

Table 2.2: Ridge Regression model parameters

Decision Tree Model

Decision trees are simple and, when charted, human-readable models. Not traditionally used in the field of bioinformatics, the Decision Tree model has been included in case it proves to be a useful yet unexplored method for the prediction of plant phenotypes. Through exhaustive grid-search cross-validation, trees were pruned by limiting the depth of the tree to eight leaf nodes, and requiring a minimum of eleven samples per leaf node. See Figure 2.3 for the full parameters of the model.

Parameter	Cross-Validation Values	Final Configuration
Criterion	Squared Error	Squared Error
Max Depth	2,4,6,8,10,15,20	8
Max Features	None	None
Max Leaf Nodes	None	None
Min Cost-Complexity Pruning	0	0
Min Impurity Decrease	0	0
Min Samples per Leaf	1,5,10,20,30,40	11
Min Samples per Split	2	2
Min Weight Fraction per Leaf	0	0
Monotonicity Constraint	None	None
Random State	611	611
Splitter	Best, Random	Best

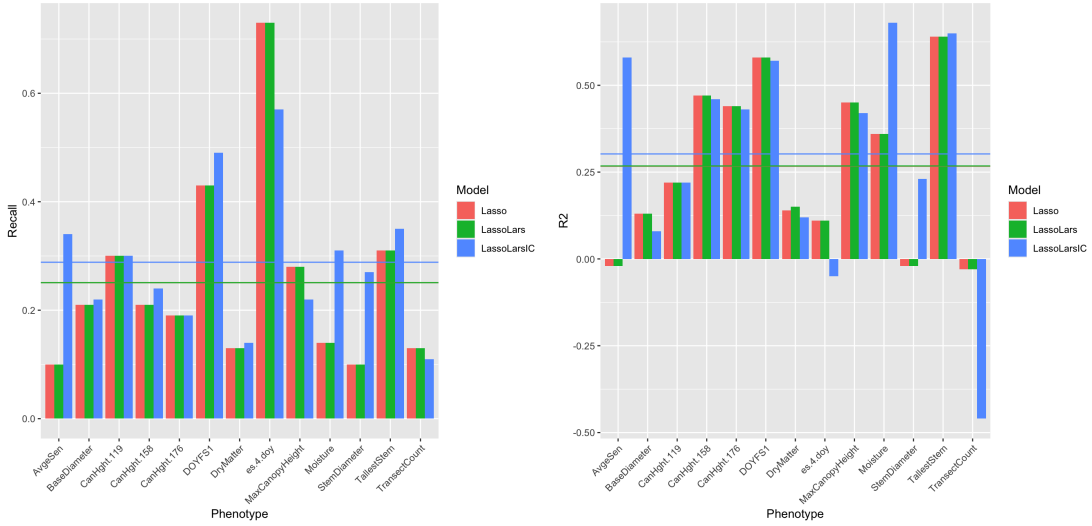
Table 2.3: Decision Tree model parameters

The Decision Tree model also makes use of the `make_tree` and `write_tree` functions from the auxiliary files. These functions take the final state of the model for each phenotype and will write a diagram of the model to a directory called `dtree` in the output directory given as an argument to PRODELS.

Least Absolute Shrinkage and Selection Operator (Lasso) Model

Least Absolute Shrinkage and Selection Operator (Lasso) models [40] have previously been used in the context of animal breeding [41–43], and have served as the foundation for the creation of new linear models [44]. Implemented in this paper using Scikit Learn, three different Lasso architectures (Lasso, LassoLars (Lasso with Least Angle Regression), LassoLarsIC (Lasso with Lars and Information-criteria) [45–47]) were

evaluated. Evaluation found that the LassoLarsIC model performed better on the whole, with an R^2 (0.27) equal to the other models but a higher mean recall rate (0.29) compared to the Lasso (0.25) and LassoLars models (0.25).



(a) (b)

Figure 2.1: The recall rate (a) and the R^2 (b) of each Lasso model by Phenotype.

The LassoLarsIC model differs from a traditional Lasso or LassoLars by making use of the Bayes Information criterion (BIC) to decide the Alpha parameter of the model. The BIC is defined by $-2\log(\hat{L}) + \log(N)d$, where N is the number of features. This is a more efficient method, computationally, for selecting the Alpha compared to a standard cross-validation method.

Parameter	Cross-Validation Values	Final Configuration
Criterion	AIC, BIC	BIC
Fit Intercept	True, False	True
Precompute	Auto	Auto
Max Iterations	500	500
Eps	2.220446049250313e-16	2.220446049250313e-16
Copy X	True	True
Positive	False	False
Noise Variance	None, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100	0.1

Table 2.4: LassoLarsIC model parameters.

Support Vector Regression (SVR) Model

Support Vector Machine (SVM) models are a cost-effective alternative to large neural networks. SVMs have previously been used to predict continuous phenotype values from Pearson ranked SNPs [48, 49]. A Radial Basis Function (RBF) kernel function, with a regularisation parameter (C) of 100 was found to be most effective at predicting the phenotype values. The regularisation parameter (C) is a squared L2 (Euclidian Length)

penalty. The strength of the regularisation performed by C is inversely proportional to the value given to the model.

Parameter	Cross-Validation Values	Final Configuration
Kernel	Linear,Poly,RBF,Sigmoid	RBF
Degree	1,3,5,10,50	N/A
Gamma	Scale,Auto	Scale
Coefficient of 0	0	N/A
Tolerance	$1e-3$	$1e-3$
C	0.001, 0.01, 0.1, 1, 10, 100, 1000	100
Epsilon	0.0001, 0.001, 0.01, 1	1
Cache Size	200	200
Max Iterations	-1	-1

Table 2.5: SVR model parameters

NuSVR Model

The NuSVR model is an alternative SVM that uses a new parameter (Nu) to determine the number of support vectors. Nu replaces the Epsilon parameter of the traditional SVR model. The NuSVR implementation in Scikit Learn is based on LibSVM [50]. Although it is not substantially different from the above SVR model, the Nu parameter may prove to make the SVM model more effective at phenotype prediction.

Parameter	Cross-Validation Values	Final Configuration
Nu	0.1, 0.3, 0.5, 0.7, 0.9	0.5
C	0.001, 0.01, 0.1, 1, 10, 100, 1000	100
Kernel	Linear, Poly, RBF, Sigmoid,	RBF
Degree	1, 3, 5, 10, 50	N/A
Gamma	Scale, Auto	Scale
Coefficient of 0	0	N/A
Shrinking	True	True
Cache Size	200	200
Max Iterations	-1	-1

Table 2.6: NuSVR model parameters

LinearSVR Model

The LinearSVR model is implemented with LibLinear [51] via Scikit Learn, rather than LibSVM. Although the model only supports a linear kernel, the difference in base architecture makes it an interesting comparison in model performance. Testing found the model performed best with an epsilon insensitive loss function, a C value of 1 and an epsilon of 0. The full details of the model can be found in Table 2.7.

Feed Forward Neural Network (FFNN)

To represent Artificial Neural Networks in general, a simple Feed Forward network was developed using Keras. The model features a 1,363 node input layer, the output of

Parameter	Cross-Validation Values	Final Configuration
Epsilon	0	0
Tolerance	1e-4	1e-4
C	0.001, 0.01, 0.1, 1, 10, 100, 1000	1
Loss	L1, L2	L1
Fit Intercept	True, False	False
Intercept Scaling	1	1
Dual	Auto	Dual
Random State	611	611
Max Iterations	1000	1000

Table 2.7: LinearSVR model parameters

which is passed through a flattening layer and a dropout layer with a rate of 0.3. This dropout repeats after each hidden layer as well. The model has two hidden layers of 64 nodes each, which pass results to a single node output. The FFNN model uses a Rectified Linear Unit (ReLU) activation function at each layer. 1,363 input nodes were used instead of 13,630, which would match the input shape of the SNP dataset, due to memory limitations of the hardware. As a value, 1,363 mirrors the binning method employed for creating confusion matrices and calculating the number of true positives, false positives, and the recall rate. 25 epochs and a batch size of 32 were used during the training of the FFNN. The FFNN model uses a mean squared error loss function. The model was optimised using the Keras implementation of the Adam algorithm [52]. Adam is first-order stochastic gradient descent method [53].

Convolution Neural Network (CVNN)

Convolution Neural Networks (CVNNs) are a popular form of ANN. The implementation of a CVNN in this paper is inspired by the Local Convolution Neural Network [54]. The input layer is a One Dimensional Conventional [55] layer of 13,630 nodes with one filter in the convolution and a convolutional window size of ten. The outputs of the convolutional input layer are flattened and passed through a dropout layer with a rate of 0.3 to the 2 hidden layers of 64 nodes each. The hidden layers also use a ReLU activation function and the output of each layer passes through a dropout layer with a rate of 0.3 before being passed to the single node output layer. The loss function for the CVNN is also the mean squared error. The model was also optimised using the Keras implementation of the Adam algorithm. The structure of this model and the FFNN can be found in Figure 2.2.

Radial Basis Function Network (RBFN)

Radial Basis Function Networks (RBFNs) have previously been shown to perform well in biocomputational research [56, 57]. Keras currently provides no Radial Basis Function (RBF) layer, however their accessible API has allowed for the creation of an RBF layer by Petra Vidnerová [58], which was modified to work with the changes made to the API and Keras backend in Keras3 [59].

This implementation of the model uses a RBF layer as the input, with random initial

centroids. A dropout of 0.2 is performed on the output of this layer before it is passed to the single node output. This output layer uses a sigmoid activation function. The model makes use of the Adam optimiser with a learning rate of 0.0001 and uses the mean squared error loss.

Due to hardware limitations, no results could be obtained from this model.

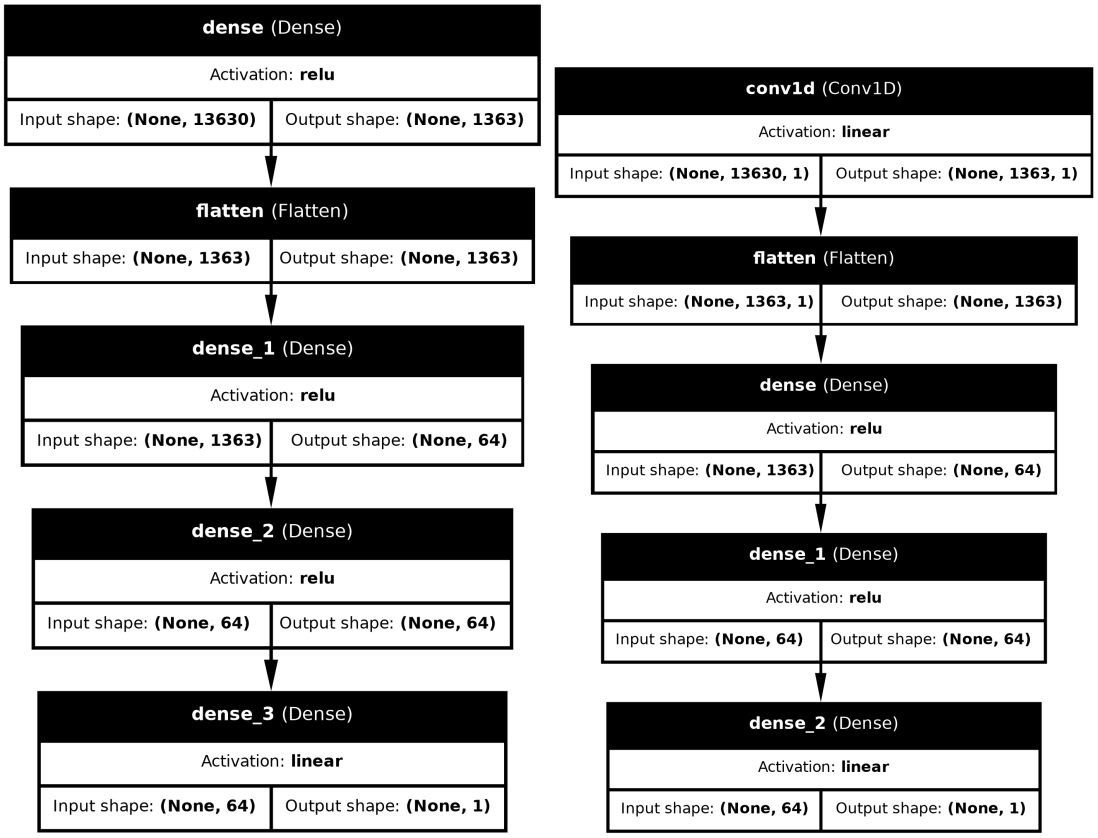


Figure 2.2: (a): Structure of the FFNN model. (b): Structure of the CVNN model.

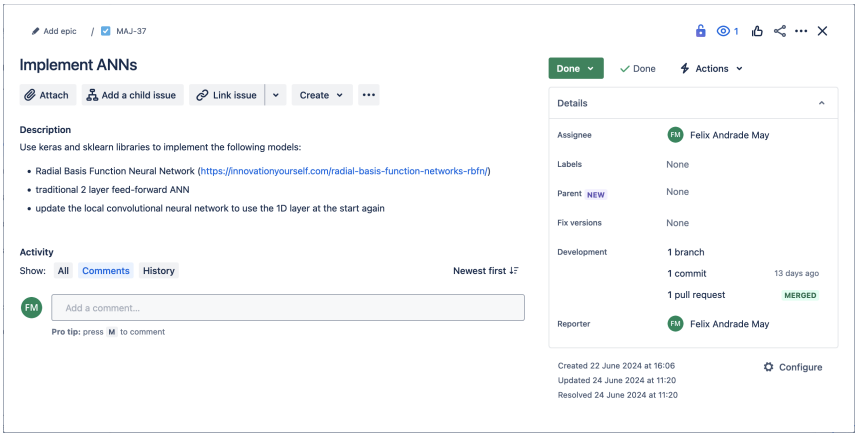
2.4 Software Development

2.4.1 Project Management

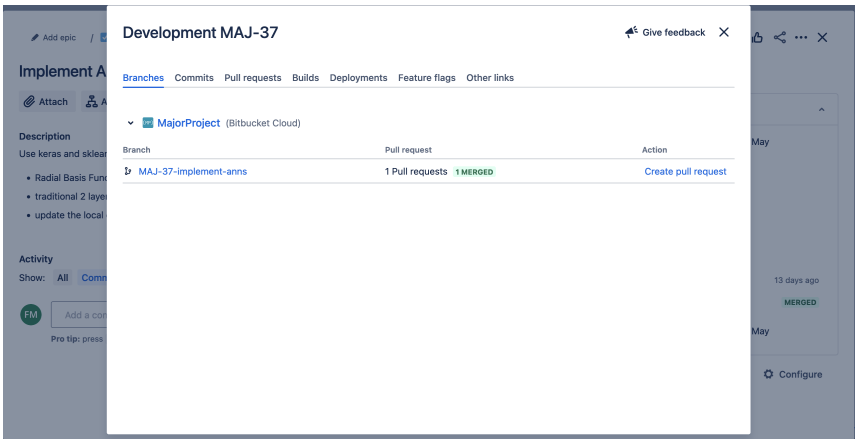
To manage the project I chose to use Atlassian's Jira and BitBucket services [13, 15]. Jira is a project management service while BitBucket is a Git-based repository hosting service. I chose to use these services as I have experience using them during my industrial year placement to develop large-scale projects. When first beginning the project I was experimenting with using GitHub, but found the integration between Jira and BitBucket was far more seamless and efficient than on GitHub.

Because both services are produced by Atlassian, they offer a wide array of integrations.

For example, when creating a work case on Jira, you can automatically link that case to a Git branch. When a case is created it is assigned an alphanumeric identifier, for this project that took the form `MAJ-000`. When creating a branch from a case, the branch is given a matching identifier, with the case name appended to the end in kebab case, so: `MAJ-000-test-functions`. Combined with BitBucket's clear pull request and code review UI the task of managing multiple open branches is made much easier.



(a)



(b)

Figure 2.3: (a): A Jira case. (b): The list of branches associated with a case.

At the CLI level, Gitk [14] was used to view the Git history of the project. Gitk visualises the commit graph and shows additional information such as author, commit message, tags, branch names, as well as the time and date of each commit. Selecting a commit shows a list of the modified files, and a preview of the changes in the file. Figure 2.4 shows the Gitk interface. The commit graph appears in the top left, with the date and author of the commit to the right. Beneath this is the file list and file difference for the selected commit. Between these fields, the selected commit's SHA 1D hash is shown, as well as a search field which can be used to search for specific lines of code in the commit history.

During the course of this project I used KanBan style boards to manage my caseload. When a case was created it would be placed into the 'To Do' category. This category of

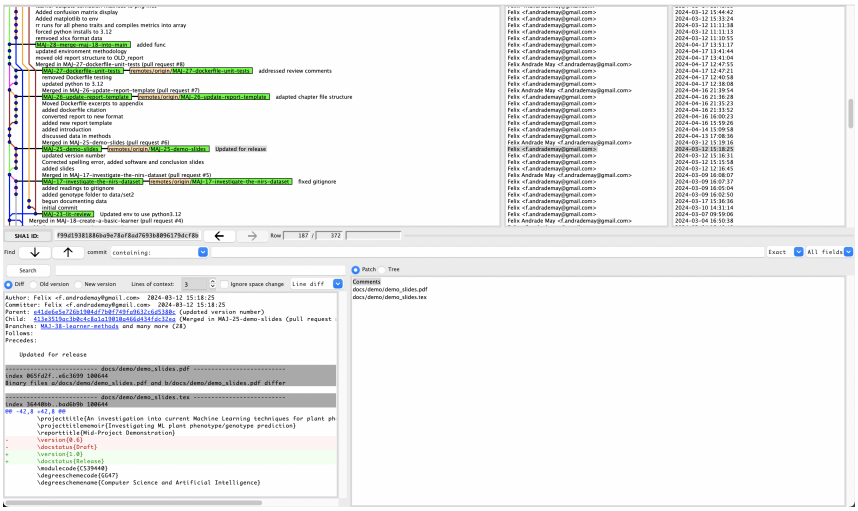


Figure 2.4: The window produced by running `gitk --all` in the CLI, showing the commit history between 04-03-2024 and 17-04-2024.

the board had no limit on the number of items. 'To Do' cases would progress to 'Planning' cases if some work had been begun on them, or an plan for how to go about the case was being created, then once work started on a case it would progress to the 'In Progress' category. Both of these categories had a maximum limit of five cases at a time. A case could be moved to the unlimited 'On Hold' category from any other category if, for example, that work had been abandoned in favour of another approach or if the case had sat in the 'Planning' or 'In Progress' category for too long. Once the branch associated with a case had been merged into the main branch of the project and closed, the case could be moved to the final 'Done' category. This workflow can be seen in Figure 2.5.

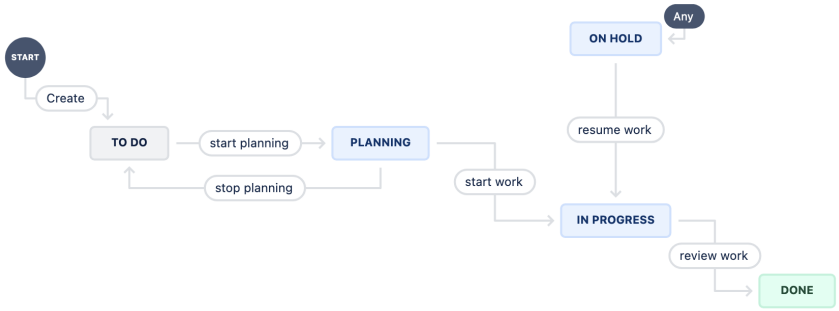


Figure 2.5: The workflow that was created for the project on Jira.

At the start of the project, a set of Epics were created outlining the key goals of the project, any case that was created would be linked to one of these Epics. The key dates of the project were then set as releases in Jira, which could be viewed through the 'Timeline' view in Jira [Figure 2.6]. This also helped to keep the workload manageable as it provided a quick way to prioritise cases by which Epic's deadline was more urgent.

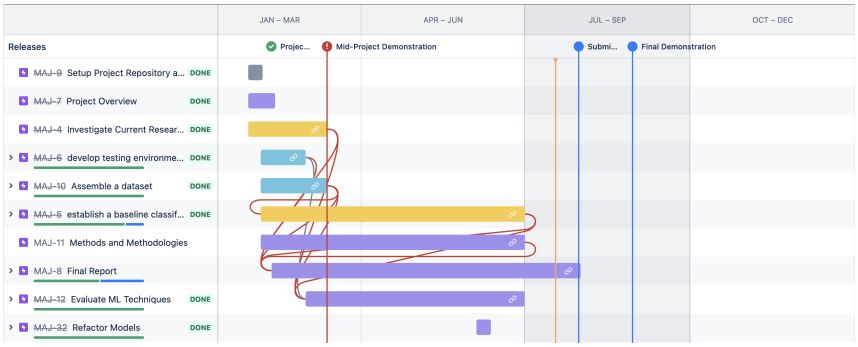


Figure 2.6: Timeline of the project's Epics.

2.4.2 Research and Software Development Approach

I had no prior experience with Scikit Learn before this project. To familiarise myself with the library I began by reading through the documentation, making use of sections of example code, to begin developing a simple model. This model served as a way for me to experiment with the syntax and the logic of Scikit Learn. Once the desired functionality had been developed into this simple model I could extrapolate the different processes into distinct functions for future use. I could then use this simple model as a guide for the development of new models, following the same pattern to simplify the development process. By simplifying the development process I was also able to make the maintenance of models easier as they all follow the same basic structure.

To select what models I used I would find papers which analysed the ability of several models, or papers which covered topics at a higher level. While reading these, I would select citations that would provide more insight into specific models or areas of the field that seemed like common knowledge.

During the development process I investigated the potential of implementing a BayesB model [10]. To do this I looked at a few different Python libraries before deciding PyMC3 [60] would be the best for this project. After creating a Gibbs sampler [61] using PyMC3 I chose to end work to implement a BayesB model using PyMC3 and Scikit Learn as I decided the time would be better spent on other cases, so it was put 'On Hold'.

Chapter 3

Results and Discussion

3.1 Results

Table 3.1 shows the highest, lowest, and mean R^2 scores for each learner. The three SVRs models returned the highest overall R^2 scores (0.75, 0.75, 0.70) when estimating the Moisture trait. Interestingly, this is also the trait that the Decision Tree, FFNN, LassoLarsIC, and Ridge Regression models received their highest R^2 from predicting. While the highest R^2 for the CVNN came from the Average Senescence category. The CVNN is the most consistent in its ability to predict phenotype values, with the lowest Standard Deviation (SD) between R^2 of any model, despite it also having one of the lowest mean R^2. This suggests that the model is consistently bad at prediction. This understanding of the CVNN's model ability is reinforced by the model also having lowest mean recall rate (0.25) [Table 3.2]. Because the R^2 can be infinitely negative, the SD of the R^2 value is larger than that of the recall rate despite many of the values being similar in their range. Despite this there is not a significant difference between the baseline model and the CVNN model which had the lowest mean ($T(24) = -0.455$; $P = 0.327$), so it can be said that all of the models are homogenous when evaluated by the R^2 metric.

Compared to the baseline Ridge Regression model, most of the models returned higher R^2 values. The Decision Tree model and the CVNN models both returned lower means and maximum R^2 values compared to the baseline [Table 3.1]. The Decision Tree model also underperformed in its mean and maximum recall rates compared to the Ridge Regression model [Table 3.2], however, the CVNN model performed similarly to the Ridge Regression model, with only a lower mean recall rate.

The recall rate, shown in Table 3.2, helps to provide a more informed understanding of the different models' performances. The SVR model still scores the highest recall (0.79), however, the highest recall rate of the Decision Tree model (0.74) is not statistically insignificant in comparison ($FMax = 1.07$). Compared to the baseline set by the Ridge Regression model all other models underperformed, with mean recall rates not significantly lower than those of the Ridge Regression model with a T-Value of $T(24) = -1.144$ and a P-Value of $P = 0.132$ as calculated between the baseline Ridge Regression model and the model with the lowest mean recall rate (CVNN at 0.25).

Model	Highest R^2	Mean R^2	Lowest R^2	SD
Ridge Regression	0.59	0.26	-0.22	0.29
Decision Tree	0.53	0.21	-0.40	0.24
LassoLarsIC	0.68	0.27	-0.46	0.34
SVR	0.75	0.28	-0.41	0.30
NuSVR	0.75	0.32	-0.12	0.27
LinearSVR	0.70	0.24	-0.54	0.36
FFNN	0.70	0.31	-0.26	0.31
CVNN	0.53	0.21	-0.21	0.20

Table 3.1: Highest, Lowest, Mean R^2 Scores, and the Standard Deviation of the R^2 for each model.

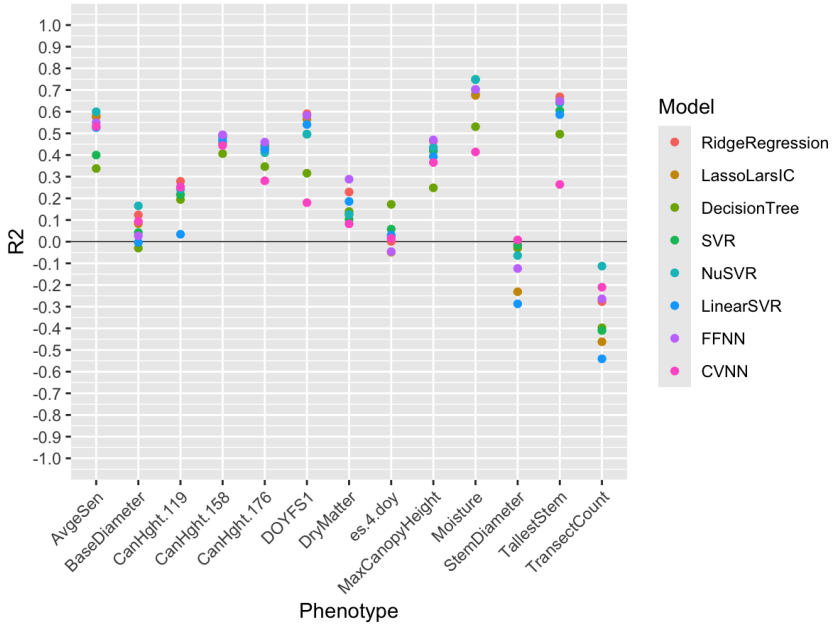


Figure 3.1: The R^2 scores for each model for each phenotype trait.

Model	Highest Recall	Mean Recall	Lowest Recall	SD
Ridge Regression	0.64	0.31	0.13	0.13
Decision Tree	0.74	0.27	0.14	0.14
LassoLarsIC	0.57	0.29	0.11	0.13
SVR	0.79	0.28	0.11	0.18
NuSVR	0.66	0.30	0.11	0.15
LinearSVR	0.53	0.28	0.09	0.12
FFNN	0.64	0.27	0.12	0.15
CVNN	0.64	0.25	0.08	0.14

Table 3.2: Highest, Lowest, Mean recall Rates, and Standard Deviation for each model.

As discussed Section 3.2, the best performing trait was the es.4.doy trait. This measures the day of the year that leaf emergence, or emergence stage four, was recorded. The

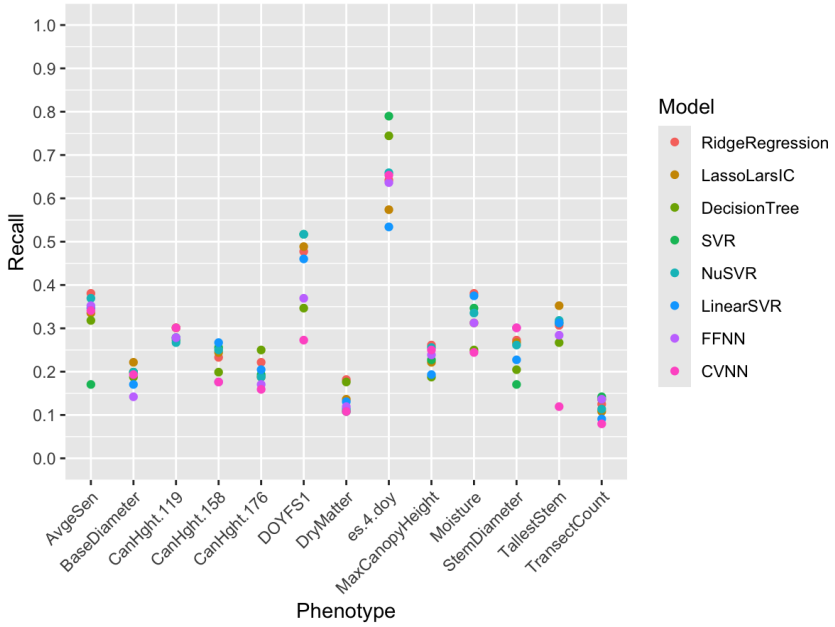


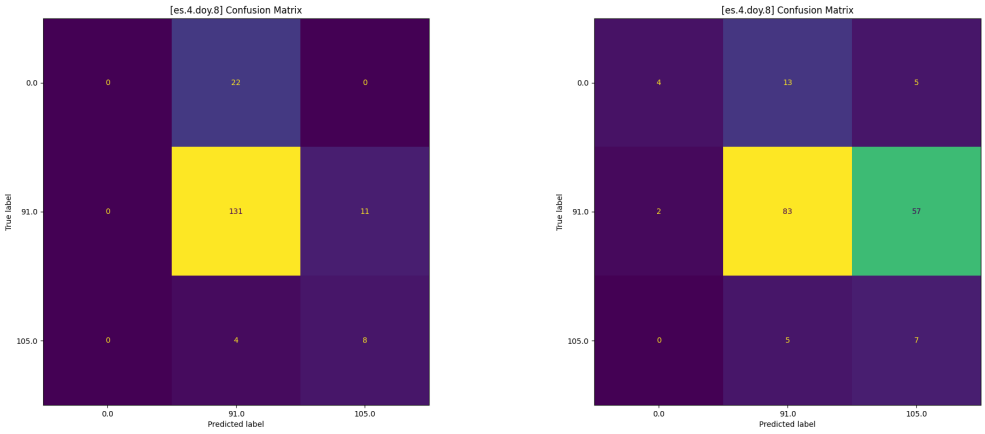
Figure 3.2: The recall rates of each model for each phenotype trait.

range of values for this trait is from 77 to 133, giving only 56 possible values. As seen in Figure 3.3, there are only three predicting values from each model, despite the number of bins used in the creation of the confusion matrices being ten for each trait. This suggests that, in addition to there being a limited range of potential values, there is also a strong correlation between the true values and the SNP configuration for each genotype.

The same pattern as shown in Figure 3.2 can be seen when inspecting the confusion matrices produced by the models. Figure 3.4 shows the confusion matrices for each models' prediction of the Dry Matter trait. This is the yield of the plant as measured in grams at the time the sample was harvested. Recall rates for this trait were amongst the lowest across all models, with the highest rate being 0.18 (Ridge Regression and Decision Tree) and a lowest recall rate of 0.11 (SVR, NuSVR, and CVNN). Figure 3.2 shows how each model tends to predict a single value, with some variation. The Ridge Regression, Decision Tree, LassoLarsIC, and LinearSVR models showed the most variation [Figures 3.4(a), 3.4(b), 3.4(c), 3.4(f)]. These four models were also the four with the highest recall rates for the trait (0.18, 0.18, 0.14, 0.13). Interestingly, the only one of these models to also have a high R^2 value is the Ridge Regression model, with an R^2 of 0.23. The FFNN model had a higher R^2 of 0.29 even though it had one of the lowest recall rates for the DryMatter trait at only 0.12. All of the confusion matrices can be found in Appendix D.

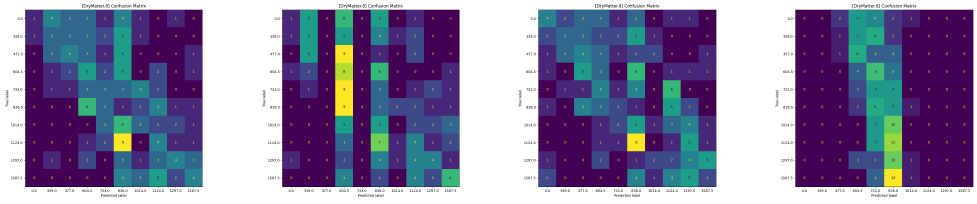
3.1.1 Average Senesence (AvgeSen)

When predicting the AvgeSen trait, the models performed similarly in both the R^2 and recall metrics, with the NuSVR and LassoLarsIC models performing the best, with an R^2

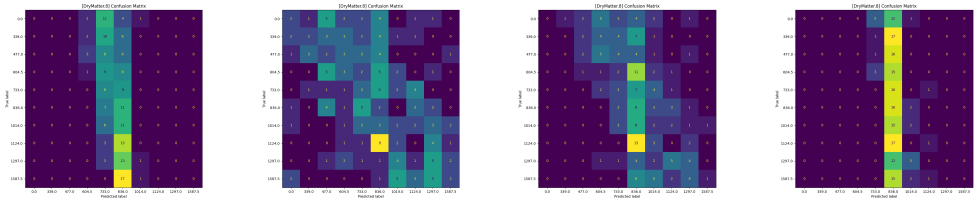


(a) (b)

Figure 3.3: The confusion matrices produced by the SVR model (a) which had the best recall rate for es.4.doy (0.79), and the LinearSVR model (b) which had the lowest recall (0.53) of any model when predicting the es.4.doy phenotype trait.



(a) Ridge Regression (b) Decision Tree (c) LassoLarsIC (d) SVR



(e) NuSVR (f) LinearSVR (g) FFNN (h) CVNN

Figure 3.4: Confusion Matrices for the DryMatter trait.

of 0.60 and 0.58 and recall rates of 0.37 and 0.34 respectively. As shown in Figure 3.5, the Decision Tree model also performed well, compared to the baseline set by the Ridge Regression model. The worst performing model for this trait was the SVR model (R^2=0.40, Recall=0.17). This is interesting as the SVR model performed on par with other models for the other twelve traits, and was the only model to perform this poorly

on the trait, as shown in Figure 3.2 and 3.5. The SVR model did perform somewhat better when measured by its R^2, however it was still the second worst performing model by this metric.

As shown in Table 3.3 the models performed similarly, though worse than the baseline in both R^2 and recall. These performances suggest that the AvgeSen has some correlation to the genetic composition, but is also influenced by the environment the plant existed within before harvest. This shows the Machine Learning models could be used to select the plants with preferable genomic compositions for future breeding, especially if combined with models trained on the relation between phenotypic traits and environmental factors.

Metric	Value
RR R^2	0.58
RR Recall	0.38
Mean R^2	0.50
Mean Recall	0.32

Table 3.3: The mean R^2 and recall values for the AvgeSen trait compared to the baseline values.

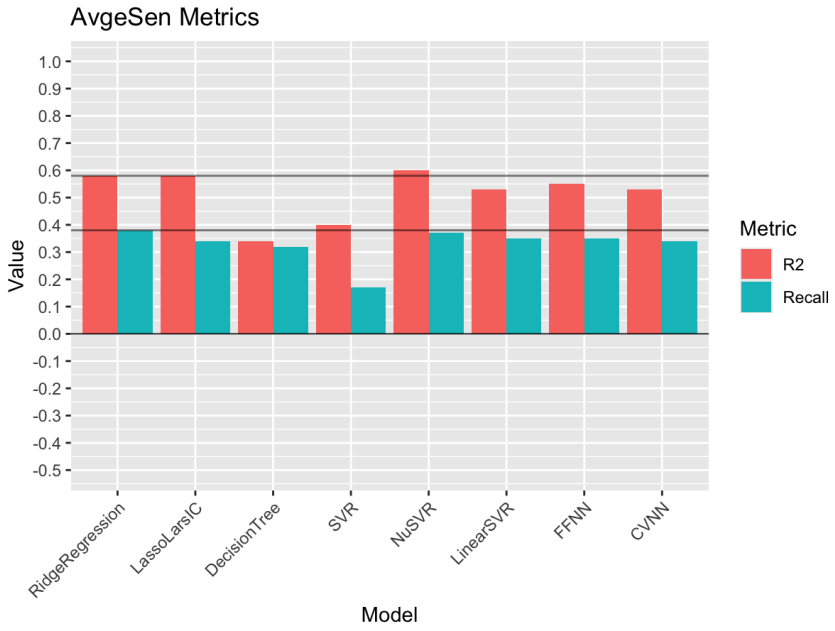


Figure 3.5: The R^2 and recall values calculated for each model when predicting the AvgeSen trait. Horizontal lines show the baseline R^2 and recall set by the Ridge Regression model.

3.1.2 Base Diameter (BaseDiameter)

The BaseDiameter trait was poorly predicted by all models with the baseline R^2 set by the Ridge Regression only reaching 0.12, the fourth lowest baseline R^2. Despite this low

score, the only model to outperform the baseline was the NuSVR model (R^2=0.16), 3.2 times the mean R^2 and 0.19 more than the lowest R^2 produced by the Decision Tree model. The models also perform poorly when measured by recall, with a baseline of 0.20, the second lowest R^2 from the Ridge Regression model. As with R^2, only the LassoLarsIC model outperformed the baseline recall rate (0.20) with a recall rate of 0.22. The SVR, NuSVR, and CVNN models all performed equally to the baseline recall rate, though the SVR and CVNN models' R^2 values were much lower than the baseline. As shown in Figure 3.6, the model that performed worst overall at predicting this trait was the LinearSVR model, which produced the second lowest R^2 (0.00) and second lowest recall rate (0.17). As discussed above the lowest R^2 score was given by the Decision Tree model (-0.03), though the model performed much better when evaluated with recall rate (0.19). The FFNN had the worst recall rate for the BaseDiameter trait at 0.14, however its R^2 was only the third worst at 0.03.

The poor performance shown by all models when predicting the BaseDiameter trait, as shown in Figure 3.6, suggests that this trait has little correlation to the genetic composition of the Miscanthus sinensis samples. This trait may be more influenced by the environmental factors, though this cannot be said for certain given the results shown here.

Metric	Value
RR R^2	0.12
RR Recall	0.20
Mean R^2	0.05
Mean Recall	0.19

Table 3.4: The mean R^2 and recall values for the BaseDiameter trait compared to the baseline values.

3.1.3 Canopy Heights

The canopy heights were measured on the 119th, 158th, 176th days of 2008. As shown in Figure 3.7, the models' ability to predict the canopy height improved past the 150 day mark, but fell somewhat when predicting the canopy height after the 175th day. When predicting the CanHght.119, the recall rates of the LassoLarsIC (0.30) and the CVNN (0.30) model match the baseline set by the Ridge Regression, however the R^2 for these models was not able to match the baseline R^2 of 0.28. The R^2 for both the ANN models performed well when compared to the baseline of 0.28, with both models having an R^2 of 0.25. Interestingly, the R^2 of the LinearSVR model is far lower than any of the other models (0.03) and does not match the pattern of ratios between the R^2 and recall rates as shown in Figure 3.7(a). This result does not repeat when predicting the CanHght.158 and CanHght.176 traits.

The mean model performance increases by a factor of 2.3 for R^2 when predicting the CanHght.158 compared to the CanHght.119, before decreasing by 0.06 when predicting the CanHght.176. This drop is mirrored by the R^2 performance of the baseline Ridge Regression, which falls from 0.49 to 0.44. Despite the changes in R^2, there is little difference in the recall rate of the models between the three traits. This suggests that the

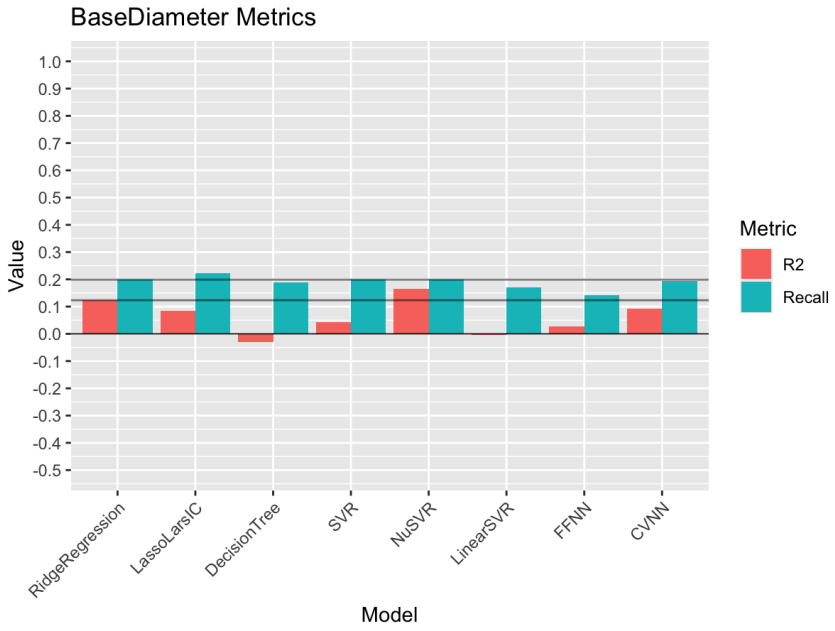


Figure 3.6: The R^2 and recall values calculated for each model when predicting the BaseDiameter trait. Horizontal lines show the baseline R^2 and recall set by the Ridge Regression model.

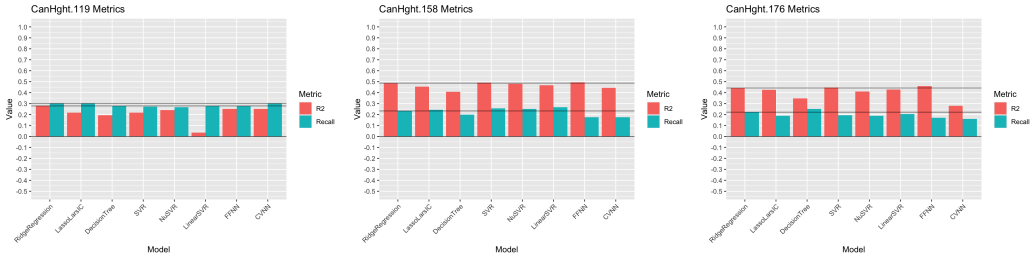
canopy height of the plants, and therefore their rate of growth, can be consistently estimated by machine learning models. It also shows that there is a consistent correlation between the genetic composition and the growth rate of Miscanthus sinensis.

Metric	CanHght.119	CanHght.158	CanHght.176
RR R^2	0.28	0.49	0.44
RR Recall	0.30	0.23	0.22
Mean R^2	0.20	0.46	0.40
Mean Recall	0.24	0.24	0.19

Table 3.5: The mean R^2 and recall values for the Canopy Height traits compared to the baseline values.

3.1.4 Day of the year that Flowering Stage One was observed (DOYFS1)

A greater variation in model performance is seen when predicting the Day of the year that Flowering Stage One was observed than has been seen in previous phenotypes, with a variation of 0.41 between the highest R^2 (Ridge Regression, 0.59) and lowest R^2 (CVNN, 0.18). There is also a variation of 0.25 between the highest recall rate (SVR & NuSVR, 0.52) and the lowest recall rate (CVNN, 0.27). As shown in Figure 3.6, the models also follow no clear pattern regarding the ratio between the R^2 and the recall rate of each model, a pattern very strongly shown in Figure 3.7(b). Inspecting the confusion matrices shown in Figure D.6, most of the values for this trait are accurately predicted as 0.0. This is the value inserted in for the models in the case of a missing



(a) CanHght.119 (b) CanHght.158 (c) CanHght.176

Figure 3.7: The R^2 and recall values calculated for each model when predicting the CanHght.119 (a), CanHght.158 (b), and CanHght.176 (c) traits. Horizontal lines show the baseline R^2 and recall set by the Ridge Regression model.

value and does not appear naturally in the phenotype dataset. With this accounted for, there is not a strong correlation between the genetic composition and the time it takes Miscanthus sinensis to flower, although with so many missing values in the original dataset it is hard to say this with any certainty.

Metric	Value
RR R^2	0.59
RR Recall	0.48
Mean R^2	0.46
Mean Recall	0.36

Table 3.6: The mean R^2 and recall values for the DOYFS1 trait compared to the baseline values.

3.1.5 Dry Matter (DryMatter)

Models struggled to predict the DryMatter trait with a low average R^2 of 0.15 and a low recall rate of 0.13, as shown by Table 3.7. The mean R^2 was not significantly different from the baseline, and neither was there a significant difference between the baseline recall rate and the mean recall rate.

With such a consistently low predictive ability, the DryMatter trait, which is the weight of the yield of the plant in grams of the sample at harvest, can be said to have no correlation with the genetic composition of the plant itself.

Metric	Value
RR R^2	0.23
RR Recall	0.18
Mean R^2	0.15
Mean Recall	0.13

Table 3.7: The mean R^2 and recall values for the DryMatter trait compared to the baseline values.

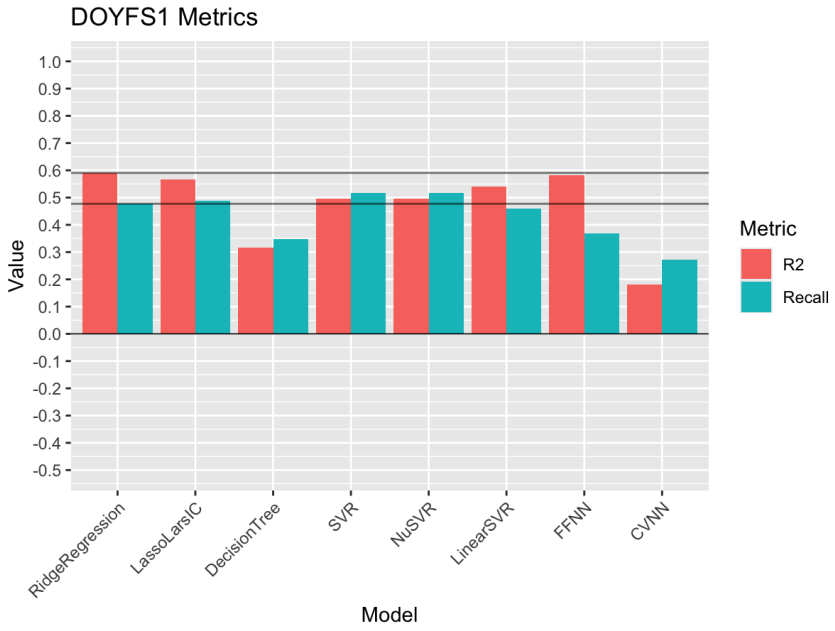


Figure 3.8: The R^2 and recall values calculated for each model when predicting the DOYFS1 trait. Horizontal lines show the baseline R^2 and recall set by the Ridge Regression model.

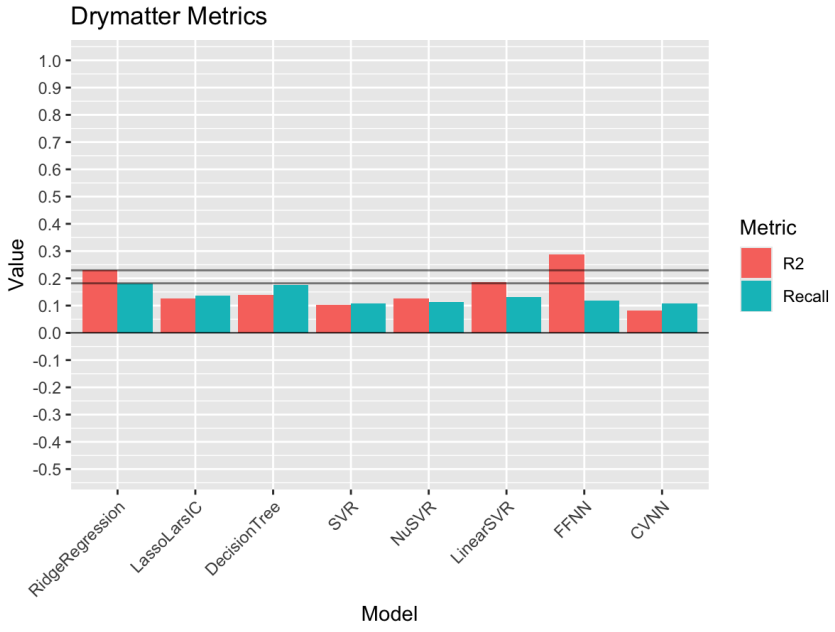


Figure 3.9: The R^2 and recall values calculated for each model when predicting the Dry-Matter trait. Horizontal lines show the baseline R^2 and recall set by the Ridge Regression model.

3.1.6 Day of the year that Emergence Stage Four was observed (es.4.doy)

The es.4.doy trait was one of the highest performance traits when scored by recall rate, with every model achieving its highest recall rate for predicting this trait. The es.4.doy measures the day of the year that leaf emergence, or emergence stage four, was recorded. The range of values for this trait is from 77 to 133, giving only 56 possible values. This small value range increases the likelihood that a model will predict the true value, or a value much closer to true compared to a trait such as TransectCount, one of the worst performing traits. This is contrasted by the low R^2, which peaked with an R^2 of 0.17 produced by the Decision Tree model, though most models produced a value marginally above 0, which was also the baseline set by the Ridge Regression model. There was no significant difference between the baseline and the mean R^2 of the models, nor was there a significant difference in the recall rates.

Metric	Value
RR R^2	0.00
RR Recall	0.64
Mean R^2	0.03
Mean Recall	0.65

Table 3.8: The mean R^2 and recall values for the es.4.doy trait compared to the baseline values.

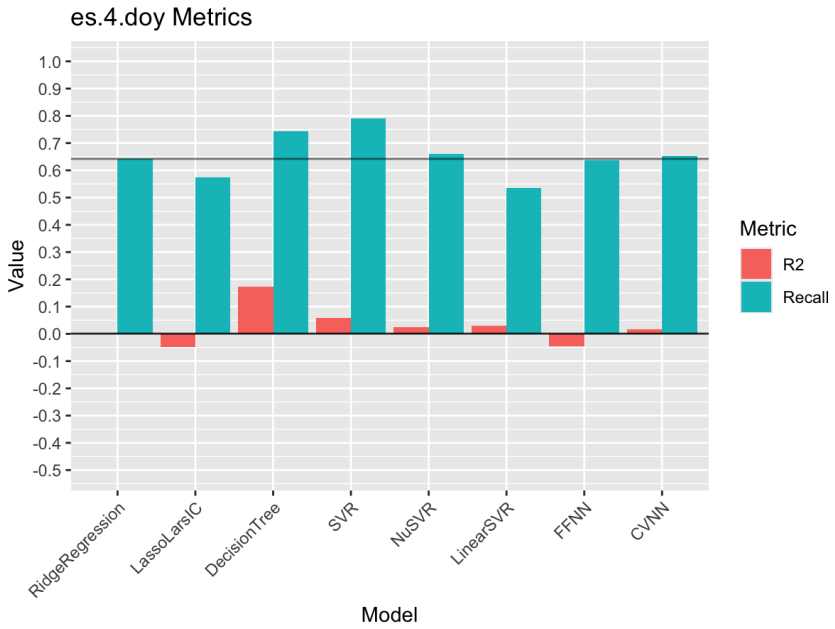


Figure 3.10: The R^2 and recall values calculated for each model when predicting the es.4.doy trait. Horizontal lines show the baseline R^2 and recall set by the Ridge Regression model.

3.1.7 Maximum Observed Canopy Height (MaxCanopyHeight)

As with the Canopy Heights discussed in Section 3.1.3, the models performed somewhat well when predicting the height of the canopy at the time of the sample's harvest. Most models scored similarly to the baseline. The Decision Tree model performed worst, with a significant decrease in ability compared to the baseline R^2 and recall rates. The only model to outperform the R^2 set by the baseline was the FFNN model, with an R^2 of 0.47. The FFNN model was only the third best model when ranked by recall rate scoring 0.24, compared to the baseline of 0.26 this is not significantly different.

Metric	Value
RR R^2	0.46
RR Recall	0.26
Mean R^2	0.34
Mean Recall	0.23

Table 3.9: The mean R^2 and recall values for the MaxCanopyHeight trait compared to the baseline values.

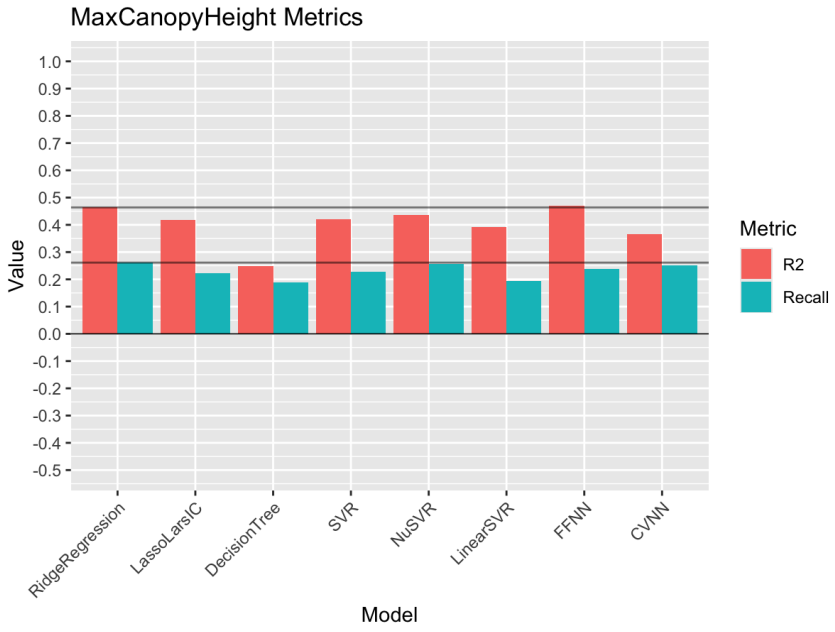


Figure 3.11: The R^2 and recall values calculated for each model when predicting the MaxCanopyHeight trait. Horizontal lines show the baseline R^2 and recall set by the Ridge Regression model.

3.1.8 Moisture (Moisture)

Models also performed well on the Moisture trait with an average R^2 of 0.65, not significantly different from the baseline R^2 of 0.68 as shown in Table 3.10. This suggests the line of regression closely matches the true position of the data, and as a result there

is strong correlation between the genomic profile of the samples and the Moisture trait. Figure 3.12 shows that only the Decision Tree and CVNN models underperformed compared to the baseline, as they have with many other traits.

Although the recall rate for the trait is only 0.31 on average, the confusion matrices [Figure D.10] show a stronger predictive ability than the recall rate alone would suggest. Combined with the high baseline and mean R^2 the moisture trait suggests some of the strongest correlation between the genome composition and phenotype. It should be noted that, unlike the es.4.doy, the Moisture trait possessed a recorded range of 0 to 52.5, which gives a possible 525 possible values (when only calculating with to the first decimal place), much more than the 56 possible values of the es.4.doy trait. Despite this difference in range the models still scored well in recall, which was higher than any individual model's mean recall rate and equal to the mean recall rate of the baseline RR model [Table 3.2].

Metric	Value
RR R^2	0.68
RR Recall	0.38
Mean R^2	0.65
Mean Recall	0.31

Table 3.10: The mean R^2 and recall values for the Moisture trait compared to the baseline values.

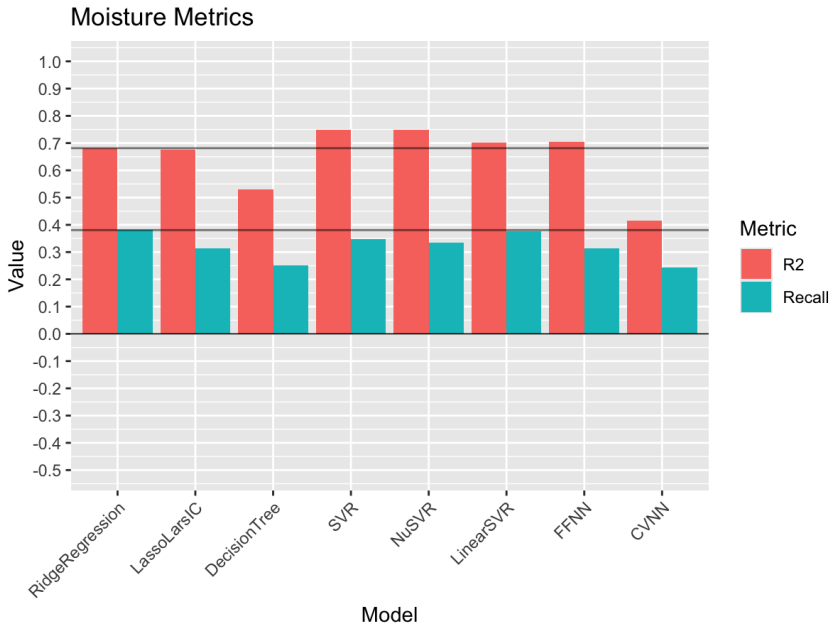


Figure 3.12: The R^2 and recall values calculated for each model when predicting the Moisture trait. Horizontal lines show the baseline R^2 and recall set by the Ridge Regression model.

3.1.9 Stem Diameter (StemDiameter)

As shown in Table 3.11 the models performed similarly to the baseline, though insignificantly worse. The recall rates for the trait are similar to those of the Moisture trait, though the confusion matrices shown in Figure D.11 show that the models had a much wider range of negative predictions, unlike what is seen with the Moisture trait. This is reflected by the poor R^2 performance of all of the models, with this trait being only two of the thirteen to have a negative R^2 baseline. As shown in Figure 3.13, the CVNN model is the best performing model overall, as it is the only model with a positive R^2 (0.01) and a recall rate (0.30) above the baseline. The FFNN model also produced a recall rate (0.30) above the baseline, however the model's R^2 (-0.12) was below both the baseline and the mean R^2 for the trait. It is interesting that the CVNN model performed well compared to the others on this trait, as the only other trait it was the best predictor of was CanHght.119, a trait which saw a much more uniform ability between the models in their predict ability.

Metric	Value
RR R^2	-0.02
RR Recall	0.27
Mean R^2	-0.10
Mean Recall	0.25

Table 3.11: The mean R^2 and recall values for the StemDiameter trait compared to the baseline values.

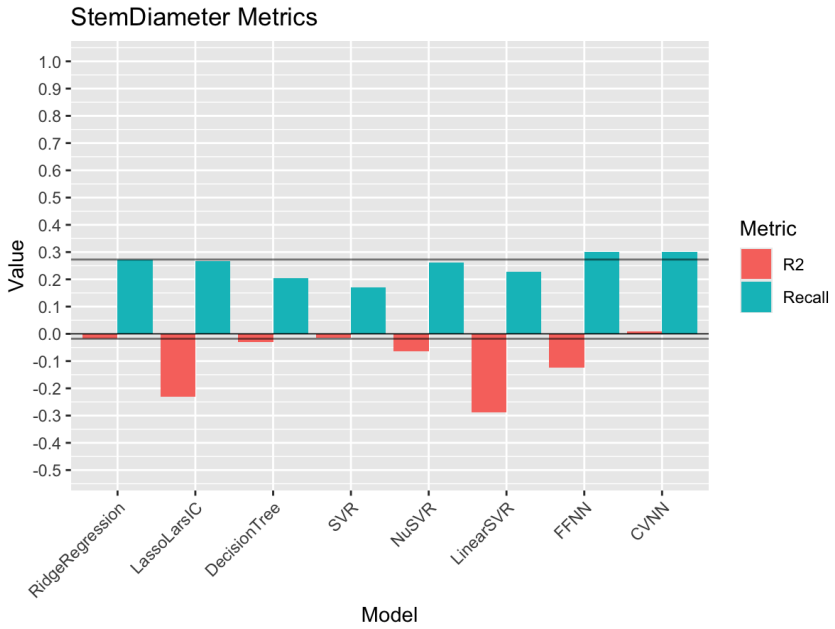


Figure 3.13: The R^2 and recall values calculated for each model when predicting the StemDiameter trait. Horizontal lines show the baseline R^2 and recall set by the Ridge Regression model.

3.1.10 Tallest Stem (TallestStem)

The TallestStem trait was the second highest performing trait by average R^2 (0.56), behind Moisture (0.65). Though models scored similar recall rates on this trait compared to most other traits, the high R^2, combined with the confusion matrices shown in Figure D.12 shows that the models are good at predicting this trait to a reasonable degree of accuracy. This suggests that there is some correlation between the genome composition and the measured phenotype trait. Though no model produced an R^2 above the baseline value, the LassoLarsIC, SVR, NuSVR, and LinearSVR models all had higher recall rates than the baseline, as shown in Table 3.12.

Metric	Value
RR R^2	0.67
RR Recall	0.31
Mean R^2	0.56
Mean Recall	0.28

Table 3.12: The mean R^2 and recall values for the TallestStem trait compared to the baseline values.

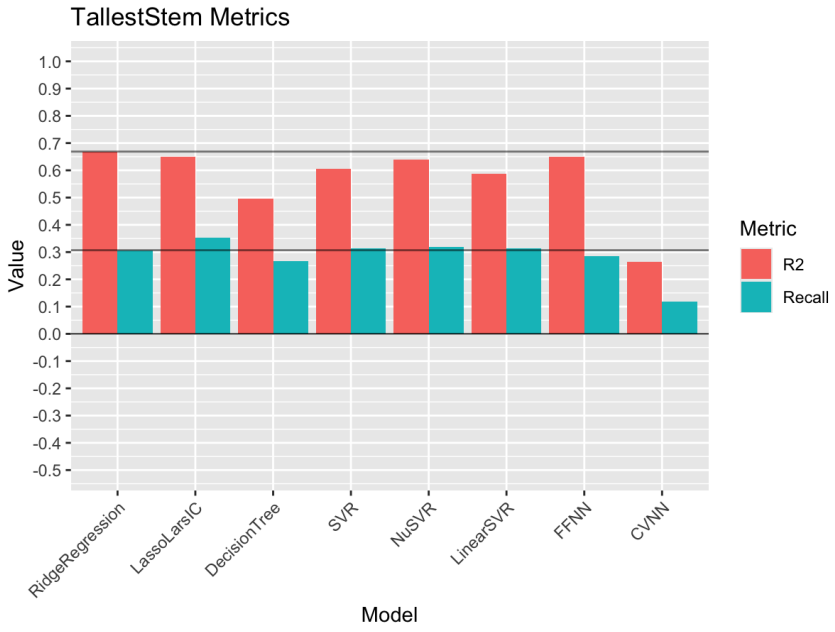


Figure 3.14: The R^2 and recall values calculated for each model when predicting the TallestStem trait. Horizontal lines show the baseline R^2 and recall set by the Ridge Regression model.

3.1.11 Transect Count (TransectCount)

The TransectCount is the worst performing trait by both R^2 and recall rate, one of only two traits with a negative baseline R^2. Models also produced consistently low R^2

values, with an average of -0.34 it was the lowest mean R^2 of any model and significantly lower than the mean and baseline R^2 of the StemDiameter trait. Both the baseline and mean recall rates are about half those of the StemDiameter trait.
Figure D.13 shows how all of the models tended towards predicting a narrow range of values for the trait compared to the total possible range of values, a tendency seen in some of the other traits with poor predictions, such as the DryMatter trait.

Metric	Value
RR R^2	-0.28
RR Recall	0.13
Mean R^2	-0.34
Mean Recall	0.12

Table 3.13: The mean R^2 and recall values for the TransectCount trait compared to the baseline values.

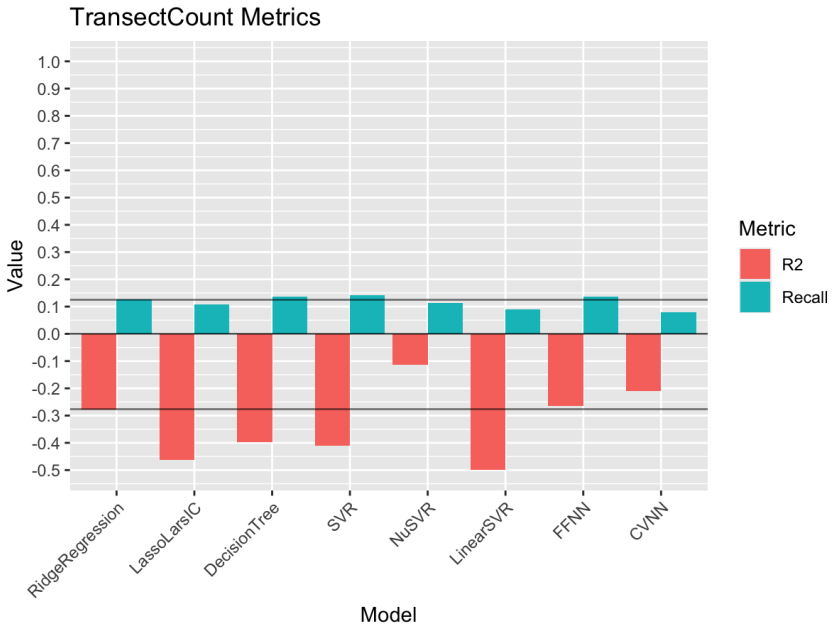


Figure 3.15: The R^2 and recall values calculated for each model when predicting the TransectCount trait. Horizontal lines show the baseline R^2 and recall set by the Ridge Regression model.

3.2 Discussion

When evaluating the models there are two aspects for considerations: how the models performed on each phenotype, and how each model performed independently across all phenotypes.

As Figure 3.2 shows, each model follows a similar pattern as regards to its predictive ability for each Phenotype trait. Every model has its highest recall rate when predicting the es.4.doy trait. As mentioned above the es.4.doy trait has a range of only 56 possible values which may have increased the predictive ability of the models by decreasing the variation in predictions. This may not be the case however, as the TransectCount trait, which contained 76 possible values, was the phenotype trait that models scored the lowest recall rates in. The TransectCount trait was determined by inserting a stick through the base of the plant and counting the number of stems that reached 50% of the canopy height which are also touching the stick. That this phenotype and the es.4.doy phenotype have such similar value ranges but such significantly different recall rates ($FMax = 12.34$), suggests that the TransectCount trait has little to do with the plants genetic construction, and far more to do with the environmental factors the plant faced in during its time growing. This would suggest that there is a strong correlation between the genomic composition of the plant and the rate at which it reaches emergence stage four, and that there is little to no correlation between the genomic composition and the number of stems that a plant will produce at its base.

If the SNPs that contribute to the rate of maturity can be identified, then this trait can be spread through a population through genetic modification processes. This would create the ability to decrease the growing times of plants, and with some consideration to other phenotype traits, not impact the yield provided by the faster maturing plants.

With the limited range of possible predictions for es.4.doy in mind, the high recall rate suggests that Machine Learnings would be useful for predicting the stages of emergence for the plant, which may be a desirable trait to select for breeding as it would help to create a uniformity amongst a crop of Miscanthus sinensis. This uniformity would simplify the harvesting process for the crop as it allows growers to breed a crop of Miscanthus that flower at a more predictable point in the growing cycle. Environmental factors such as the amount of sunlight and the temperature will still contribute to this, though a combined genomic and environmental model would account for the variation introduced by environmental factors.

In the case of biofuel production, the Dry Matter content of a plant, combined with the Moisture content of a plant would give an indication of the oil content of the plant. These triacylglycerol oils produced by plants are an important part in the production of biofuels as they are one of the most abundant forms of concentrated carbon available in plant matter [62]. If we are able to understand the connection between the composition of a plants genome and the potential oil content, then the ability to selectively breed plants with a lower moisture content, and a higher oil content would be increased. From a post-harvest point of view, knowing the weight of a plant and the genetic composition would allow biofuel producers to discard plants that may not produce enough biofuel mass given their harvested content. If a crop is harvested that has a genome prone to creating plants with low oil production and high water retention, then these crops may

be discarded before the biofuel production begins. This would save energy and production time as the need to extract and filter out the water content of the plant would no longer be needed. This would also decrease the emissions produced through the creation of biofuels as less processing is required, and therefore less energy needed. Further study should be conducted regarding this, preferably with the oil content of the plant included as a measured phenotype.

The Maximum Observed Canopy Height trait would be a desirable trait for selective breeding as the taller the crop the more susceptible it may be to strong winds and storms which would damage a crop and decrease the yield. By being able to predict the height of a crop from the genetic composition growers could selectively breed crops of Miscanthus sinensis to suit their growing climate. The Maximum Observed Canopy Height could also be combined with other phenotype traits such as the DryMatter to breed plants that spend less of their time producing large amounts of flowers and seeds and more growing longer and sturdier stems. This would be beneficial in the conversion of the crops to biofuels as more useable plant would be produced. If breeding models were also used to predict the canopy heights at different stages, plants with a genome that produces fast growing stems could be favoured in the breeding process to create a strain of Miscanthus sinensis which grows taller, faster, therefore creating more resources that could be turned into biofuel in a shorter span of time, which again increases the quantity of creatable biofuel.

Though combined the models performed generally well, with a mean R^2 of 0.26 and a mean recall rate of 0.28 when predicting quantitative values split into an average of nine bins, no individual model was the best performing outright. The NuSVR model performed the best on the most models (four of thirteen), however the Ridge Regression model also performed best on the same number of traits as the NuSVR model. The other SVM learners, SVR and LinearSVR, performed well on three traits together. Combined with the results of the NuSVR model, the SVM models were the most successful style of Machine Learning. The performance of the Ridge Regression model should not be discounted though. As the baseline model the Ridge Regression consistently outperformed other more complex models, whether or not the trait in question was correlated to the genome or not.

The models examined in this paper can be compared to prior research [63,64]. Models in this paper performed performed similarly to the ordinary least squared regression model examined by Arruda et al, which found marker-assisted models had a performance ability of < 0.3 while the models in this paper had a mean recall rate of 0.28. It should be noted, however, that the actual evaluation metrics employed by Arruda et al. are not entirely comparable to those used in this paper. Albrecht et al., however, found that models with a similar input sample size to that used in this paper performed somewhat better than models in this paper (0.43 - 0.53).

The CVNN model which was inspired by [54], and the RBFN which was inspired by [56], were not fully replicated in this paper for a variety of reasons discussed in Section 4.4. Further research with improved resources should be conducted to evaluate these models more accurately, as the previous research indicates these methods should perform better than has been demonstrated here. An expanded dataset, with a larger number of genotypes, should also be conducted. Increasing the range of inputs would

not only provide more learning material for the models, but it may also mirror industrial applications of these models more closely. This is important if ML models are to be used in the industrial genomic selection of plants for hybrid breeding for improved biofuel production. A dataset of phenotypes which includes more complex phenotypes, such as a measure of triacylglycerol oils in the plant would also be useful in showing the ability of ML models to predict complex phenotypes from Miscanthus sinensis genotypes.

3.3 Conclusion

Machine Learning models, in particular Support Vector Regression like the NuSVR model and linear models such as Ridge Regression or Best Linear Unbiased Predictor, are capable of somewhat accurately predicting quantitative phenotype traits from the genomic composition of the Miscanthus sinensis grass. The models were able to show some correlation between the plant genome and the resulting phenotypes, the TallestStem, Moisture, AvgeSen, and DOYFS1 were the best predicted traits and can therefore be said to be most strongly correlated to the Miscanthus sinensis genome. The performance of the models could be improved if the environmental factors are taken into consideration alongside an expanded set of genomic information, though further research should be conducted to see if this hypothesis holds.

Chapter 4

Evaluation and Reflection

During the course of this project I was met with a variety of challenges and obstacles to overcome. These ranged from minor confusions to major roadblocks. I was, however, able to overcome them at each step through considered planning and a strong work ethic. This chapter will discuss these problems and how I overcame them in more detail. It will also evaluate the methods and techniques used throughout the project and discuss if and how they could have been improved or altered.

4.1 Data Acquisition

At the beginning of the project, during the initial research phase, I had planned to use a dataset derived from one or several research papers. I had chosen this approach as I thought using existing, published, material would also allow me to compare the ability of my Machine Learning methods to those from the original papers because they were being trained on the same set of data. This proved to be harder than I had expected as many of the papers I came across initially made little to no mention of the content of their data. If the data was discussed in any detail, only the fact that it was simulated data was discussed, or the online site hosting the data had been shutdown or simply no longer existed on the internet, these latter situations were disappointingly but not surprisingly common. I had decided it was not in the scope of this research to simulate the data because I did not think it would allow for an accurate assessment of the correlation between the plant genome and the phenotypes. After discussing this with my supervisor, Prof. Reyer Zwiggelaar, I was connected with a research group at the Prifysgol Aberystwyth who provided me with access to their Miscanthus sinensis datasets. At first, I was provided with an archive that contained a large mix of Comma-separated Value datasets, R source files, proprietary Microsoft Excel files, and text files. It took me some time to sort through the files and begin to understand what I had been handed. After further discussion via email with the group I was sent another archive with more CSV files and more R source files, again with no explanation of the content of the archive. Reading back through the email chain I had been linked into, and which contained the prior correspondence, I learned the group met fortnightly over the Microsoft Teams teleconferencing software. After asking to attend a meeting I was

invited to their next meeting which would be on 28th March, 2024. At the meeting I was able to clarify some confusion (again) as to the fact I was completing and undergraduate project, and not a PhD. I was also able to learn some more about the datasets that I had, although most of this knowledge came not from the meeting itself but from a GitHub page that I was given access to. This contained the data that I already had but in a sorted and explained manner that meant I could actually begin on implementing models to make use of the appropriate data. Over the following weeks I had some additional questions that were helpfully answered by James Strong (Prifysgol Aberystwyth) and a Stephen Gow (University of Southampton).

The challenge of acquiring data took far longer than I had originally estimated it would take. This delayed the project immensely as without any data I could not begin developing any ML models, and without these I could not reasonably begin documenting much of the methodology described in Chapter 2. I could also not begin writing any of Chapter 3. This would not have been so much of an issue if I had been able to compile the data sooner, but by the time the task was completed it was mid April and the original time frame for the project as a whole was drawing close. This reduction in time lead to some of the design aspects described in the paper. The models all follow a very similar internal design, with minor variations to account for the style of each model where necessary. This was done to simplify the design process and to reduce the time spent developing each model. This allowed for more time to be spent running the models and tuning the models' parameters. It also increased the amount of time available for the production of this paper.

If I were to reproduce this research, or conduct further research with this paper as a foundation, there are several approaches I would take to improve my access to data. Primarily, I would begin by looking only for open-source datasets. After I had began to use the Miscanthus sinensis dataset seen in this paper I came across other research, such as [48], which made use of publicly available datasets from organisations such as the Rice Diversity Panel [65]. This would not only have shortened the process of data acquisition, but it would have also allowed me to compare the results of this research to other papers that have also made use of one of the Rice Diversity Panel's data. If I had instead continued to use the Miscanthus sinensis dataset, I would have been, and in the future I will be, able to ask more informed questions to the suppliers of the data. These questions would have, I expect, returned more useful answers than the ones I actually received, which would have furthered my understanding of the dataset much faster. With my current understanding of the field and the data itself, I would have been more confident in my approach to contacting the suppliers. As a result of this confidence I would have, and will be in the future, more proactive in trying to organise face-to-face meetings to ask questions and have the data described to me. This too would have furthered my understanding of the data faster than the limited online meeting I was able to attend during the project.

4.2 Software Development

The problem of data acquisition did, however, show the resiliency of the software development methods, discussed in Chapter2, that I had established at the start of the

project. I was able to easily keep on top of work because of the KanBan boards. The hard limit of five cases for the 'Planning' and 'In Progress' categories, combined with the Jira timeline view [Figure 2.6], meant I was able to prioritise the epics and cases that were more urgent. This was further aided by the integration between Jira and BitBucket, as I was able to easily manage the different branches I was working in. It also made the process of merging branches together simpler because of the Pull Request process in BitBucket. This allowed me to manually review each changed file in a branch compared to the file as it was in the destination branch. If there was something that needed changing, such as additional empty lines or blocks of commented out test code that I had not removed, I could add a review comment to that line in BitBucket. This comment would need to be marked as 'Resolved' before I could approve and then merge the branch into the destination branch. After I had reviewed each file I would return back to the development branch on my machine, make the necessary changes, commit and push the changes to the branch and then re-review the branch in BitBucket. If there were no other changes that needed to be made, then the pull request would be approved and then merged into the destination branch by BitBucket. If in the case of a merge conflict BitBucket would identify these before allowing the branches to be merged. These conflicts would still need to be updated in the local branch and the changes pushed to the origin branch. I found this process to be far less error-prone compared to managing the version control process without the assistance of BitBucket, as well as it being simpler and easier with BitBucket and Jira.

By reducing the version control workload with BitBucket and Jira, I was able to devote more time to expanding my technical abilities. I have previously created short and simple bash scripts for use on personal projects but also in professional scenarios such as during my industrial placement year. These bash scripts were only normally used to automate tedious CLI processes and would only contain a few commands to be run sequentially by the script. To expand the scope of the technical work for this project, I chose to create an executable bash script that could be used within the development environment to run a desired module with the correct CLI parameters. This required me to learn a lot about the bash scripting language that I had not previously needed to interact with. A good example of this was the implementation of argument flags to the program. I previously understood how bash handled variables both from the CLI and between different scripts at a surface level. However, to implement the flags seen in Listing B.4, I had to teach myself about the 'getopts' command and the accompanying 'OPTARGS' variable. I did this by using some examples provided on StackExchange [66, 67], as well as through a few linux-centric blogs such as [68]. By reading these pages, and others, I was able to better understand not only how CLI flags work, but also how to create better, more complex and capable, bash scripts. The PRODELS script that resulted from this work greatly simplified the process of running the models as I no longer needed to enter the CLI arguments in the same order every time I needed to run a model. Instead the User is now able to use the CLI flags to organise their arguments, which can be entered into the CLI in any order. The bash script will then run the given Python file with the arguments in the correct order every time. I also investigated if there was a proper formatting for the help messages that most CLI programs provide. Although I could find no definitive guide I was able to put together a help dialogue by looking at the format of programs I have installed on my personal machine as well as from some answers online [69].

4.3 The Python and R Languages

Due to the overlap between the fields of computer science and the bioinformatics field, a lot of code is written in the R language [28]. Before this project I had some surface level experience with R, mainly to create graphs for other papers, however I needed to have a much deeper understanding of R in order to complete this project. R shares some similarities with other languages I am familiar with like Python and Haskell [70,71], though it also contains syntax that I have not had to interact with or understand before. As mentioned above, many of the files I was sent as part of the original Miscanthus datasets were written in R, this challenged me as I did not fully understand the processes that were being executed within the file. In order to expand my understanding I began by reading through the comments left in the various R files that I thought might be useful to me. I combined the information in the comments with my general understanding of programming and software development to infer the function of each individual line in the files, piecing together my understanding of the code along the way. I found this approach worked well for me, even if at times it may have been too slow it was not so important as I was not planning on doing any major development in R at the start of the project. While planning and evaluating my approaches to certain other problems during the development process I did consider using R to solve the problem at hand, though each time I deemed Python to be an easier and faster approach because I have much more experience in the language, which was important given the concerns regarding the shortening time frame mentioned above. I did use R, along with the ggplot2 package [72] to produce the graphs shown throughout the paper.

The most obvious disadvantage to disregarding the R language as a possible platform is that it also prevented me from using many packages for predictive ML models that are commonly utilised in bioinformatics. These packages include the rrBLUP [73]library whioch implements linear ridge regression models for genomic predictions. These packages would have saved some development time, and would have allowed me to examine the performance of commonly employed models, rather than analogous models that I did implement. However, the time saved by using these R packages would have not made up for the time that would have been needed for me to familiarise myself with R well enough to accomplish the task. I also ran into several problems installing R into the container that was used for development and testing. Pursuing this course would have exhausted yet more time installing R into the container along with all of the R packages, which cannot be installed as simply into a system as third party python libraries can.

In order to implement the models into Python, I chose to use several libraries I had no previous first hand experience with, namely Pandas, Scikit Learn, PyMC3, and Keras3 [19–21,60,70]. As discussed in Chapter 2, I created an example Python file to teach myself the basic syntax of Scikit Learn, to reference during further development, and to add experimental features to before placing them into the model being developed. I followed the same process when learning how to use PyMC3 and Keras3, as I found it was a simple and effective method for learning the basic logic of these libraries. For Pandas, however, I used the CLI Python environment, invoked with the `python3` command. By doing this inside of the container I was able to rerun the same commands several times over in order to understand the resulting outputs. I also found

it very useful as after each modification to the Pandas DataFrame I was able to print the DataFrame to the CLI and inspect it more easily than if I was to develop an entire program first and print out the result of each command at each step and then scroll back through the output history in the CLI.

4.4 Model Improvements

These packages allowed me to produce nine models in total, eight of which I was able to run. I was also able to produce several experimental models and scripts that I did not discuss in this paper. The Convolution Neural Network model, which was consistently one of the worst performing models, could have been improved and fleshed out far more than it is at the end of this project. Inspired by the Locally Connected Neural Network (LCNN) developed in [54], I was not able to completely recreate the model for a few reasons, not only because because the `LocallyConnected1D` layer that is the focus of the model was depreciated between Keras2 and Keras3. The Python code for the original LCNN was made available by the paper, though it was hard to find as the names used for supplementary files in the paper did not match the name of the supplementary files available online. I could not, however, find the dataset used by the paper for training the LCNN. Despite having cloned and updated the appropriate files from the Keras2 to make the `LocallyConnected1D` layer functional in Keras3 again, not being able to access the original data made it harder to implement a one-to-one recreation of the model in Keras3 because I could not fully understand the structure of the LCNN model without knowing the inputs. This was compounded by the somewhat unnecessarily complex structure of the paper's code, which made it harder for me to parse as an interested reader.

The RBF model that was mentioned briefly in Chapter 2 was another model directly inspired by a paper, only in this case the model fell short for other reasons. After forking the original RBF layer [58] and updating it to work with the new Keras3 backend [59], the layer requires an input shape equal to the shape of the input data. This causes a problem as the input shape of the SNP dataset is 13,630 by 110. When the model was run, Linux would kill the process after around ten seconds to prevent the process from using all of operating system's, and therefore the container's, memory. I attempted to solve this problem in a number of ways. Firstly, I tried increasing the memory allotted to the container from 4GB to 6GB, this did not work. I also tried running the program on my desktop machine. This machine runs Linux and has 46GB of memory, after adjusting the memory allocation for the container on this machine I was still unable to run the model without it being killed by Linux. While running it on the desktop machine I also tried splitting the data into equal subsets and training the model on each subset separately before combining the predictions at the end to get average scores for each subset model. This also did not work so the model was abandoned, finished but un-executable, and the Jira case moved to the 'On Hold' category.

4.5 Reflection

Working on this project has taught me a lot about research oriented software development, a field I am certainly more interested in now than I was six months ago. It has given me space to read papers and investigate topics that I would not necessarily have thought to delve into independently, and in doing so has helped me to expand my understanding of the field of computational bioinformatics as well as strengthen my understanding of Machine Learning as an applied practice. I also more fully understand the benefits of certain agile methodologies, having been able to apply and adapt them to my own work as opposed to being funnelled through the agile processes as I sometimes experienced during my industrial year. With this experience to hand, I certainly intend to make use of more agile processes in future work for my own personal projects, and if the situation arises, I will suggested appropriate agile processes to team leaders in industry.

References

[1] J. Huang, J. Yang, S. Msangi, S. Rozelle, and A. Weersink, "Biofuels and the poor: Global impact pathways of biofuels on agricultural markets," *Food policy*, vol. 37, no. 4, pp. 439–451, 2012.

[2] W. E. Tyner, "The us ethanol and biofuels boom: its origins, current status, and future prospects," *BioScience*, vol. 58, no. 7, pp. 646–653, 2008.

[3] J. R. Stewart, Y. Toma, F. G. Fernandez, A. Nishiwaki, T. Yamada, and G. Bollero, "The ecology and agronomy of miscanthus sinensis, a species important to bioenergy crop development, in its native range in japan: a review," *Gcb Bioenergy*, vol. 1, no. 2, pp. 126–153, 2009.

[4] M. F. Danilevicz, M. Gill, R. Anderson, J. Batley, M. Bennamoun, P. E. Bayer, and D. Edwards, "Plant genotype to phenotype prediction using machine learning," *Frontiers in Genetics*, vol. 13, p. 822173, 2022.

[5] T. Guo and X. Li, "Machine learning for predicting phenotype from genotype and environment," *Current Opinion in Biotechnology*, vol. 79, p. 102853, 2023.

[6] Y. Cui, R. Li, G. Li, F. Zhang, T. Zhu, Q. Zhang, J. Ali, Z. Li, and S. Xu, "Hybrid breeding of rice via genomic selection," *Plant biotechnology journal*, vol. 18, no. 1, pp. 57–67, 2020.

[7] R. K. Varshney, A. Graner, and M. E. Sorrells, "Genomics-assisted breeding for crop improvement," *Trends in plant science*, vol. 10, no. 12, pp. 621–630, 2005.

[8] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT press, 2018.

[9] F.-W. Li and A. Harkess, "A guide to sequence your favorite plant genomes," *Applications in Plant Sciences*, vol. 6, no. 3, p. e1030, 2018.

[10] T. H. Meuwissen, B. J. Hayes, and M. Goddard, "Prediction of total genetic value using genome-wide dense marker maps," *genetics*, vol. 157, no. 4, pp. 1819–1829, 2001.

[11] J. Wang, Z. Zhou, Z. Zhang, H. Li, D. Liu, Q. Zhang, P. J. Bradbury, E. S. Buckler, and Z. Zhang, "Expanding the blup alphabet for genomic prediction adaptable to the genetic architectures of complex traits," *Heredity*, vol. 121, no. 6, pp. 648–662, 2018. [Online]. Available: https://doi.org/10.1038/s41437-018-0075-0

[12] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for agile software development," 2001. [Online]. Available: https://agilemanifesto.org/

[13] Atlassian. (2024) Bitbucket. [Online]. Available: https://www.atlassian.com/software/bitbucket

[14] Git. (2024) Gitk. [Online]. Available: https://git-scm.com/docs/gitk

[15] Atlassian. (2024) Jira. [Online]. Available: https://www.atlassian.com/software/jira

[16] G. Slavov, P. Robson, E. Jensen, E. Hodgson, K. Farrar, G. Allison, S. Hawkins, S. Thomas-Jones, X.-F. Ma, L. Huang, *et al.*, "Contrasting geographic patterns of genetic variation for molecular markers vs. phenotypic traits in the energy grass miscanthus sinensis," *GCB Bioenergy*, vol. 5, no. 5, pp. 562–571, 2013.

[17] G. T. Slavov, R. Nipper, P. Robson, K. Farrar, G. G. Allison, M. Bosch, J. C. Clifton-Brown, I. S. Donnison, and E. Jensen, "Genome-wide association studies and prediction of 17 traits related to phenology, biomass and cell wall composition in the energy grass miscanthus sinensis," *New phytologist*, vol. 201, no. 4, pp. 1227–1239, 2014.

[18] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[20] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[21] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[22] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56 – 61.

[23] Docker. Docker desktop: The #1 containerization software for developers and teams. [Online]. Available: https://www.docker.com/products/docker-desktop/

[24] R. Mefi. (2019) Unit testing writing dockerfiles like a software developer. [Online]. Available: https://medium.com/@renatomefi/unit-testing-writing-dockerfiles-like-a-software-developer-1759f416ce84

[25] G. Vitta. (2020) Docker unit test: how to test a dockerfile (guide 2020). [Online]. Available: https://www.gasparevitta.com/posts/docker-unit-test-dockerfile-image/

[26] G. Dubinka, bodax, and rultor, "docker-unittests," https://github.com/dgroup/docker-unittests, 2019.

[27] Acsbidoul, Dstufft, Pf_moore, Pradyunsg, Uranusj, and Xafer. (2024) pip 24.0. [Online]. Available: https://pip.pypa.io/en/stable/

[28] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2023. [Online]. Available: https://www.R-project.org/

[29] H. Krekel, B. Oliveira, R. Pfannschmidt, F. Bruynooghe, B. Laugher, and F. Bruhin, "pytest 8.2.2," 2004. [Online]. Available: https://github.com/pytest-dev/pytest

[30] Pandas Contributor Community, "DataFrame," *Intro to data structures*, 2024. [Online]. Available: https://pandas.pydata.org/docs/user_guide/dsintro.html#basics-dataframe

[31] A. Amor, L. Liu, and Y. Xiao. (2024) Gridsearchcv - scikit-learn 1.5.1. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[32] ——, "R score, the coefficient of determination," *Model Selection and Evaluation*, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/model_evaluation.html#r2-score

[33] ——. (2024) sklearn.metrics - scikit-learn 1.5.1. [Online]. Available: https://scikit-learn.org/stable/api/sklearn.metrics.html

[34] ——. (2024) Ridge - scikit-learn 1.5.1. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

[35] G. Moser, B. Tier, R. E. Crump, M. S. Khatkar, and H. W. Raadsma, "A comparison of five methods to predict genomic breeding values of dairy bulls from genome-wide snp markers," *Genetics Selection Evolution*, vol. 41, pp. 1–16, 2009.

[36] M. Pszczola, T. Strabel, A. Wolc, S. Mucha, and M. Szydlowski, "Comparison of analyses of the qtlmas xiv common dataset. i: genomic selection," in *BMC proceedings*, vol. 5. Springer, 2011, pp. 1–5.

[37] T. Luan, J. A. Woolliams, S. Lien, M. Kent, M. Svendsen, and T. H. Meuwissen, "The accuracy of genomic selection in norwegian red cattle assessed by cross-validation," *Genetics*, vol. 183, no. 3, pp. 1119–1126, 2009.

[38] P. Le Roy, O. Filangi, O. Demeure, and J.-M. Elsen, "Comparison of analyses of the xv th qtlmas common dataset iii: Genomic estimations of breeding values," in *BMC proceedings*, vol. 6. Springer, 2012, pp. 1–6.

[39] N. Heslot, H.-P. Yang, M. E. Sorrells, and J.-L. Jannink, "Genomic selection in plant breeding: a comparison of models," *Crop science*, vol. 52, no. 1, pp. 146–160, 2012.

[40] R. Tibshirani, "Regression Shrinkage and Selection Via the Lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 12 2018. [Online]. Available: https://doi.org/10.1111/j.2517-6161.1996.tb02080.x

[41] G. de los Campos, H. Naya, D. Gianola, J. Crossa, A. Legarra, E. Manfredi, K. Weigel, and J. M. Cotes, "Predicting Quantitative Traits With Regression Models for Dense Molecular Markers and Pedigree," *Genetics*, vol. 182, no. 1, pp. 375–385, 05 2009. [Online]. Available: https://doi.org/10.1534/genetics.109.101501

[42] M. G. Usai, M. E. Goddard, and B. J. Hayes, "Lasso with cross-validation for genomic selection," *Genetics Research*, vol. 91, no. 6, p. 427436, 2009.

[43] K. A. Weigel, G. de los Campos, O. Gonzlez-Recio, H. Naya, X. L. Wu, N. Long, G. J. M. Rosa, and D. Gianola, "Predictive ability of direct genomic values for lifetime net merit of holstein sires using selected subsets of single nucleotide polymorphism markers," *Journal of Dairy Science*, vol. 92, pp. 5248–5257, 2009.

[44] H. Li, J. Wang, and Z. Bao, "A novel genomic selection method combining gblup and lasso," *Genetica*, vol. 143, no. 3, pp. 299–304, 2015. [Online]. Available: https://doi.org/10.1007/s10709-015-9826-5

[45] A. Amor, L. Liu, and Y. Xiao, "Lasso," *Linear Models - scikit-learn 1.5.1 Documentation*, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/linear_model.html#lasso

[46] ——, "LassoLars," *Linear Models - scikit-learn 1.5.1 Documentation*, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/linear_model.html#least-angle-regression

[47] ——, "LassoLarsIC," *Linear Models - scikit-learn 1.5.1 Documentation*, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/linear_model.html#lasso-lars-ic

[48] A. Aljouie and U. Roshan, "Prediction of continuous phenotypes in mouse, fly, and rice genome wide association studies with support vector regression snps and ridge regression classifier," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2015, pp. 1246–1250.

[49] N. Long, D. Gianola, G. J. Rosa, and K. A. Weigel, "Application of support vector regression to genome-assisted prediction of quantitative traits," *Theoretical and applied genetics*, vol. 123, pp. 1065–1074, 2011.

[50] C. Chang and C. Lin. (2024) Libsvm – a library for support vector machines. [Online]. Available: https://www.csie.ntu.edu.tw/~cjlin/libsvm/

[51] L. Chih-Jen, X. Wang, K. Chang, C. Hsieh, R. Fan, G. Yuan, F. Huang, C. Ho, B. Chu, M. Lee, W. Chiang, C. Hsia, Y. Zhu, H. Yu, H. Huang, J. Hsia, H. Chou, P. Lin, C. Lee, L. Gallo, J. Yen, Y. Li, and G. Chen. (2024) Liblinear – a library for large linear classification. Machine Learning Group at National Taiwan University. [Online]. Available: https://www.csie.ntu.edu.tw/~cjlin/liblinear/

[52] F. Chollet *et al.* (2024) Adam. [Online]. Available: https://keras.io/api/optimizers/adam/

[53] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. [Online]. Available: https://arxiv.org/abs/1412.6980

[54] T. Pook, J. Freudenthal, A. Korte, and H. Simianer, "Using local convolutional neural networks for genomic prediction," *Frontiers in genetics*, vol. 11, p. 561497, 2020.

[55] F. Chollet *et al.* (2024) Convolutional1d. [Online]. Available: https://keras.io/api/layers/convolution_layers/convolution1d/

[56] J. González-Camacho, G. de Los Campos, P. Pérez, D. Gianola, J. Cairns, G. Mahuku, R. Babu, and J. Crossa, "Genome-enabled prediction of genetic values using radial basis function neural networks," *Theoretical and Applied Genetics*, vol. 125, pp. 759–771, 2012.

[57] G. Morota and D. Gianola, "Kernel-based whole-genome prediction of complex traits: a review," *Frontiers in genetics*, vol. 5, p. 363, 2014.

[58] P. Vidnerová. (2019) Rbf-keras: an rbf layer for keras library. [Online]. Available: https://github.com/PetraVidnerova/rbf_keras

[59] F. Andrade May. (2024) Rbf-keras: an rbf layer for keras library. [Online]. Available: https://github.com/Aoianai/rbf_keras

[60] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck, "Probabilistic programming in python using pymc3," *PeerJ Computer Science*, vol. 2, p. e55, 2016.

[61] G. Casella and E. I. George, "Explaining the gibbs sampler," *The American Statistician*, vol. 46, no. 3, pp. 167–174, 1992.

[62] T. P. Durrett, C. Benning, and J. Ohlrogge, "Plant triacylglycerols as feedstocks for the production of biofuels," *The Plant Journal*, vol. 54, no. 4, pp. 593–607, 2008.

[63] M. Arruda, A. Lipka, P. Brown, A. Krill, C. Thurber, G. Brown-Guedira, Y. Dong, B. Foresman, and F. Kolb, "Comparing genomic selection and marker-assisted selection for fusarium head blight resistance in wheat (triticum aestivum l.)," *Molecular Breeding*, vol. 36, pp. 1–11, 2016.

[64] T. Albrecht, V. Wimmer, H.-J. Auinger, M. Erbe, C. Knaak, M. Ouzunova, H. Simianer, and C.-C. Schön, "Genome-based prediction of testcross values in maize," *Theoretical and Applied Genetics*, vol. 123, pp. 339–350, 2011.

[65] R. D. Panel. (2018) Rice diversity panel website. [Online]. Available: http://www.ricediversity.org/outreach/course/index.cfm

[66] gagneet, codeforester, V. Vagabond, vaeVictis, A. Requate, TomRoche, B. Karwin, jones77, and et al. (2008) Using getopts to process long and short command line options. [Online]. Available: https://stackoverflow.com/questions/402377/using-getopts-to-process-long-and-short-command-line-options

[67] R. Samane, P. T, and D. Williamson. (2012) bash getopts with multiple and mandatory options. [Online]. Available: https://stackoverflow.com/questions/11279423/bash-getopts-with-multiple-and-mandatory-options

[68] Z. Islam Laku. (2024) How to Use getopts in Bash [Complete Guide]. [Online]. Available: https://linuxsimply.com/bash-scripting-tutorial/functions/script-argument/bash-getopts/

[69] Yifan, desernaut, davetron5000, hellow, L. Finley, S. Wing, MotherDawg, Alex, pmr, J. Harley, A. Unkrig, C. J. John, and shelter. (2012) Is there a "standard" format for command line/shell help text? [Online]. Available: https://stackoverflow.com/questions/9725675/is-there-a-standard-format-for-command-line-shell-help-text

[70] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009, 1441412697.

[71] S. Marlow *et al.*, "Haskell 2010 language report," 2010. [Online]. Available: http://www.haskell.org/

[72] H. Wickham, *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. [Online]. Available: https://ggplot2.tidyverse.org 978-3-319-24277-4.

[73] J. B. Endelman, "Ridge regression and other kernels for genomic selection with r package rrblup," *Plant Genome*, vol. 4, pp. 250–255, 2011.

[74] J. Bailey-Serres, J. E. Parker, E. A. Ainsworth, G. E. Oldroyd, and J. I. Schroeder, "Genetic strategies for improving crop yields," *Nature*, vol. 575, no. 7781, pp. 109–118, 2019.

[75] P. Pérez, G. de Los Campos, J. Crossa, and D. Gianola, "Genomic-enabled prediction based on molecular markers and pedigree using the bayesian linear regression package in r," *The plant genome*, vol. 3, no. 2, 2010.

[76] P. Robson, M. Mos, J. Clifton-Brown, and I. Donnison, "Phenotypic variation in senescence in miscanthus: towards optimising biomass quality and quantity," *Bioenergy Research*, vol. 5, pp. 95–105, 2012.

[77] C. L. Davey, P. Robson, S. Hawkins, K. Farrar, J. C. Clifton-Brown, I. S. Donnison, and G. T. Slavov, "Genetic relationships between spring emergence, canopy phenology, and biomass yield increase the accuracy of genomic prediction in miscanthus," *Journal of Experimental Botany*, vol. 68, no. 18, pp. 5093–5102, 2017.

[78] J. Yan and X. Wang, "Machine learning bridges omics sciences and plant breeding," *Trends in Plant Science*, vol. 28, no. 2, pp. 199–210, 2023.

[79] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[80] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sept. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[81] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[82] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving,

M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

Appendices

Appendix A

Use of Third-Party Code, Libraries and Generative AI

1.1 Third Party Code and Software Libraries

1.1.1 Third Party Software

The Docker software [23] was used for the creation and deployment of Docker Images and Containers. It was used without modification under the Docker Subscription Service Agreement and the Docker Terms of Service.

The Python3 programming language [70] was used for the development of ML models. It was used without modification under the PSF License Agreement.

1.1.2 Python Libraries

The Keras3 library [20] was used for the development of the FFNN, CVNN, and RBFN models. It was used without modification under the Apache 2.0 license.

The Matplotlib library [79] was used to create diagrammatic depictions of models and results. It was used without modification under the License agreement for matplotlib versions 1.3.0 and later.

The NumPy library [80] was used for handling numerical data. It was used without modification under the NumPy License.

The OpenCV library for Python [81] was used for reading and handling images in tests. It was used without modification under the Apache 2.0 license.

The Pandas library [30] was used for reading CSV files and storing data. It was used without modification under the BSD 3-Clause licence.

The Pytest library [29] was used for testing Python language utility files. It was used without modification under the MIT license.

The Scikit Learn library [19] was used for the development and evaluation of ML models. It was used without modification under the BSD 3-Clause license.

The TensorFlow library [82] was used in the development of ANNs. It was used without modification under the Apache 2.0 license.

The rbf_keras layer [58] was used in the development of the RBFN model. It was used with modification [59] under the MIT license.

1.2 Generative AI

No Generative AI tools have been used for this work.

Appendix B

Code Excerpts

2.1 Docker

```
FROM ubuntu:noble
LABEL author=fea6@aber.ac.uk

WORKDIR /environment

RUN apt-get update -y && apt-get upgrade -y

# install packages
RUN apt-get install -y \
    build-essential \
    graphviz \
    libncurses5-dev \
    libgdbm-dev \
    libnss3-dev \
    libssl-dev \
    libreadline-dev \
    libffi-dev \
    pip \
    pkg-config \
    python3.12 \
    python3.12-venv \
    tmux \
    vim \
    wget \
    zlib1g-dev

# set python venv env
ENV VIRTUAL_ENV=/opt/venv
RUN python3 -m venv $VIRTUAL_ENV
ENV PATH="$VIRTUAL_ENV/bin:$PATH"

# install python libraries
# opencv 4.10.0.84 has python import errors and is unusable - 2024-08-06
RUN pip install \
    keras \
    matplotlib \
    numpy \
```

```
    opencv-python-headless==4.10.0.82 \
    pandas \
    pytest \
    scikit-learn \
    tensorflow

# copy in and make prodels exec
COPY ./prodels.sh /environment/prodels
RUN chmod +x /environment/prodels

COPY ./run_tests.sh /environment/run_tests
RUN chmod +x /environment/run_tests

RUN export PATH="$PATH:/environment/prodels"
RUN export PATH="$PATH:/environment/run_tests"
RUN export PATH="$PATH:/environment/src/models/"
```

Listing B.1: `Dockerfile` contains the start-up process for the Docker Image.

```
#!/bin/bash
version=19

cd "$(dirname "$0")"

starting_dir=$(pwd)

cd ./src/env/
# to save on system space, and to ensure the new container's name is not reserved, remove all old containers
docker container prune --force
# build the image
docker build -t testing_environment . && docker tag testing_environment testing_environment:$version

cd $starting_dir

# start container from image
docker run -it --env=Display -v ./src/:/environment/src/ -v ./data/:/environment/data/ testing_environment:$version /bin/bash
```

Listing B.2: `start_container.sh` prepares and runs the environment container.

```
docker cp <container_name>:/environment/outputs/model_results/ /major_project/my_results/
```

Listing B.3: An example command to copy files from the container to the host machine. This example copies the contents of the `model_results` directory within the container to the `major_projectmy_results/` directory.

2.2 Scripts

```
#!/bin/bash

version=0.7.1

# go to the current directory
cd "$(dirname "$0")"
```

```
7
8  # establish the flags
9  model_f=''
10 pheno_f=''
11 snp_f=''
12 output_f=''
13 debug_f=0
14
15 # print the help dialogue
16 print_usage() {
17   printf "PRODELS - PROject moDELS $version"
18   printf "\n\nUsage: prodels [options] [file ..]  runs a given model on given
       datasets"
19   printf "\n\nArguments:"
20   printf "\n    -d {0|1},            0 by default runs the model in debug mode
        if set to 1"
21   printf "\n    -h,                  displays this helpful message :)"
22   printf "\n    -m (with file name),  the name of the Model to train"
23   printf "\n    -o (with file name),  the name of the directory with /
       environment/outputs/ to write results to"
24   printf "\n    -p (with file name),  the full path of the Phenotype CSV
       dataset"
25   printf "\n    -s (with file name),  the full path of the SNP CSV dataset"
26   printf "\n    -v,                  displays the program version"
27   printf "\n"
28   exit 1
29 }
30
31 # print the version dialogue
32 print_version() {
33     printf "PRODELS - PROject moDELS $version"
34     printf "\n"
35     exit 1
36 }
37
38 # get the flag args
39 while getopts 'hvm:p:s:o:d:' flag; do
40   case "${flag}" in
41     h) print_usage ;;
42     v) print_version ;;
43     m) model_f="${OPTARG}" ;;
44     p) pheno_f="${OPTARG}" ;;
45     s) snp_f="${OPTARG}" ;;
46     o) output_f="${OPTARG}" ;;
47     d) debug_f="${OPTARG}" ;;
48     *) print_usage
49        exit 1 ;;
50   esac
51 done
52
53 # run model, pheno file, snp file, output file
54 time python3 $model_f $pheno_f $snp_f $output_f $debug_f
55
56 printf "######################      DONE      ######################"
57 printf "\n"
```

Listing B.4: `prodels` copied into the container - it's used to run models from the CLI with flagged arguments.

2.3 Python

```
import os
import sys
import csv
import pandas as pd

def remove_non2TT_genos(si8_path, snp_path):
    pheno_df = pd.read_csv(si8_path)
    snp_df = pd.read_csv(snp_path)
    print(pheno_df)
    print(snp_df)

    filtered_df = snp_df[snp_df['geno'].isin(pheno_df['geno'])]
    print(snp_df)

    snp_filename, snp_extension = os.path.splitext(os.path.basename(snp_path))
    outfile = snp_filename + "_filtered" + snp_extension
    filtered_df.to_csv(outfile, index=False)

if __name__ == "__main__":
    si8_path = sys.argv[1]
    snp_path = sys.argv[2]

    remove_non2TT_genos(si8_path, snp_path)
```

Listing B.5: `snp_snipper.py` removes rows from a Pandas DataFrame that are not present in another before saving the result to a new 'filtered' file.

Appendix C

Phenotype Dataset Description

Table C.1: Description of each column in the phenotype dataset. Compiled from previous research [16, 17, 76, 77].

Name	Description	Data Type	Values	Example	Information Type
geno	genotype	String	Mb followed by ID	”Mb-10”	support
block	There are four replicate blocks, each genotype occurs once in each	Int	1 - 4 (inc)	1,2,3,4	support
FieldOrder	TODO	int	Range 2-1000	30, 816, 1000	support
row	Each block is laid-out as a row by column grid. This is the plants row location on the grid.	int	Range 1 - 40 (inc)	2,6,11	support
col	Each block is laid-out as a row by column grid. This is the plants column location on the grid.	int	Range 1 - 25 (inc)	2,6,11	support
species	Species of the plant	string	Categories {sinensis}	”sinensis”	support
plate.id	Plate identification number when genotyping	string	Plate + tag	”Plate1B10”	support
order	TODO	int	1 - 142 (inc)	38	support

Table C.1: Description of each column in the phenotype dataset. Compiled from previous research [16, 17, 76, 77].

Name	Description	Data Type	Values	Example	Information Type
pop	Hierarchical population genetic structure (sin1_1 Continent, sin1_2 Japan)	string	Categories {sin1_1, sin1_2}	"sin1_1"	support
long	Longitude where plant was originally collected.	float	124.852 - 140.74	140.74	support
lat	Latitude where plant was originally collected.	float	32.6697 - 41.2628	38.4005	support
alt	Altitudinal data (metres above sea level)	float	30 - 1320	38.4005	support
in.138	TODO	int	single data value	1	
baseDiameter.8	The diameter measured at ground level across the widest part of the plant's base (mm)	int	150 - 700	450, 500	trait
TransectCount.8	Number of stems across the middle of the plant, determined by inserting a stick through the base and counting the stems touching the stick (that reach 50% of canopy height).	int	2 - 78	12, 31	trait
TallestStem.8	The length of the tallest stem from the base to the uppermost ligule, measured before harvest (cm)	int	47 - 265	187, 223	trait

Table C.1: Description of each column in the phenotype dataset. Compiled from previous research [16, 17, 76, 77].

Name	Description	Data Type	Values	Example	Information Type
MaxCanopy Height.8	Maximum canopy height recorded (cm)	int	35 - 205	140,130	trait
StemDiameter.8	Diameter measured approximately 10-15 cm from the base of the plant on a randomly chosen stem (mm)	float	2-9	5.5,8	trait
Moisture.8	The percentage of water in the sample	float (%)	0 - 52.45902	27.8048	trait
DryMatter.8	Yeild of plant (g)	int	5 - 3440	1147,660	trait
DOYFS1.8	Day Of Flowering Stage 1 refers to the first observable indication of flowering, recorded as the day of the year when the first flag leaf emerged	int	0-365	128,220	trait

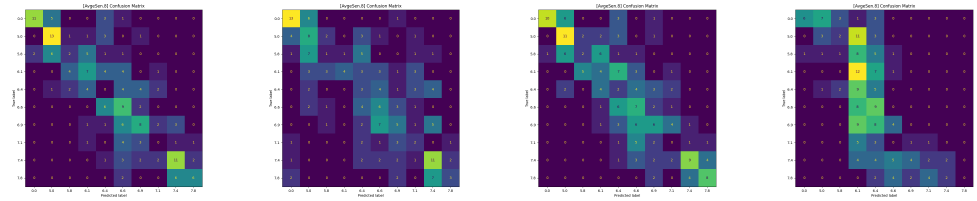
Table C.1: Description of each column in the phenotype dataset. Compiled from previous research [16, 17, 76, 77].

Name	Description	Data Type	Values	Example	Information Type
AvgeSen.8	Senescence score (0 is no senescence, 10 is all green leaf area lost) Observations were made every 2-3 weeks throughout the growing season, and AvgeSen was calculated as the average of all senescence scores for each plant over the approximately 4 months prior to harvest (Sept - Jan)	float	theoretical 0-10, actual(3.6 - 8.6)	6.4, 7	trait
es.4.doy.8	The day of the year that emergence score 4 (leaf emergence) is recorded	int	77 - 133	105,119	trait
CanHght.119.8	Canopy height (in cms) on day 119, 2008	int	0 - 52	30,35	trait
CanHght.158.8	Canopy height (in cms) on day 158, 2008	int	0 - 125	85,100	trait
CanHght.176.8	Canopy height (in cms) on day 176, 2008	int	0 - 145	100,115	trait

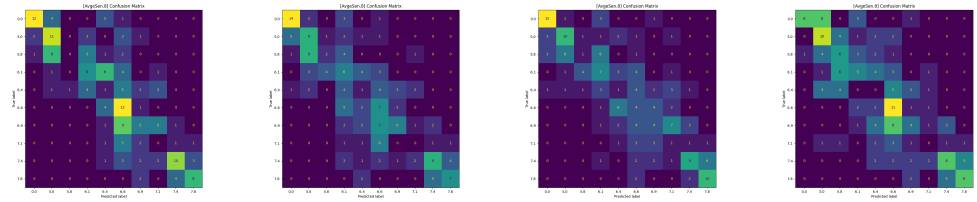
Appendix D

Model Confusion Matrices by Phenotype

A confusion matrix was created from each model for each phenotype. The continuous predictions were binned into ten groups, or fewer if ten was not possible as is the case for the 'es.4.doy' trait.

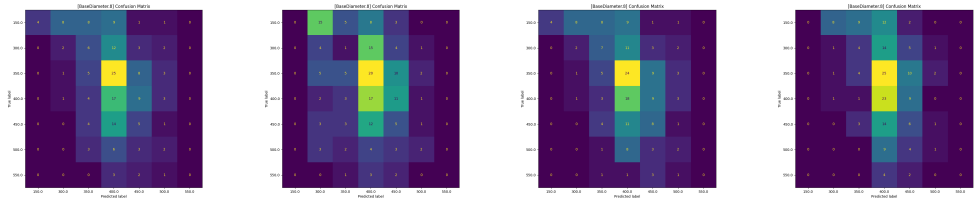


(a) Ridge Regression (b) Decision Tree (c) LassoLarsIC (d) SVR

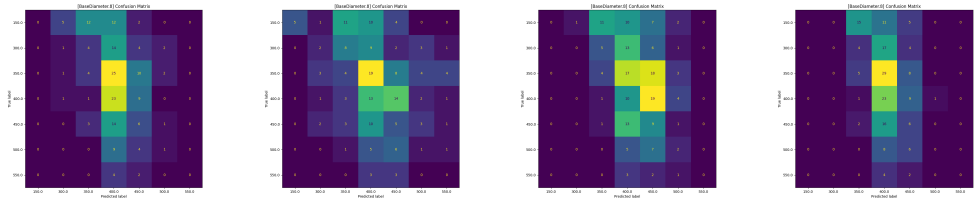


(e) NuSvr (f) LinearSVR (g) FFNN (h) CVNN

Figure D.1: Confusion Matrices for the AvgeSen trait.

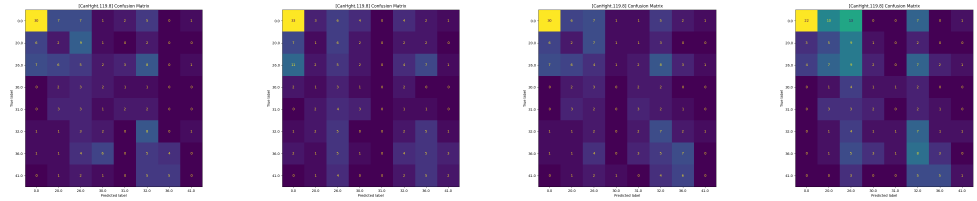


(a) Ridge Regression (b) Decision Tree (c) LassoLarsIC (d) SVR

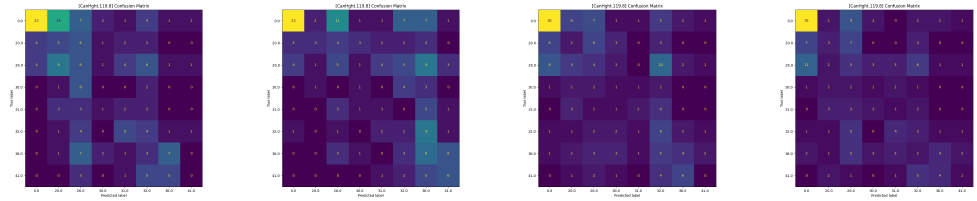


(e) NuSvr (f) LinearSVR (g) FFNN (h) CVNN

Figure D.2: Confusion Matrices for the BaseDiameter trait.

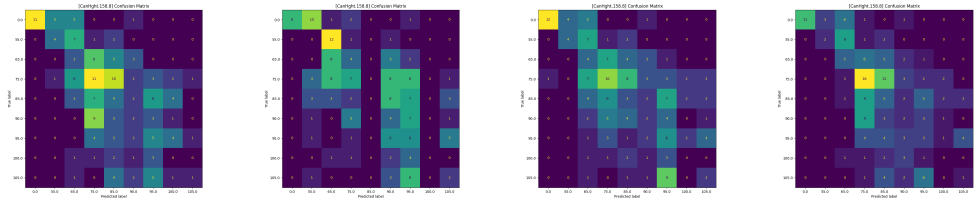


(a) Ridge Regression (b) Decision Tree (c) LassoLarsIC (d) SVR

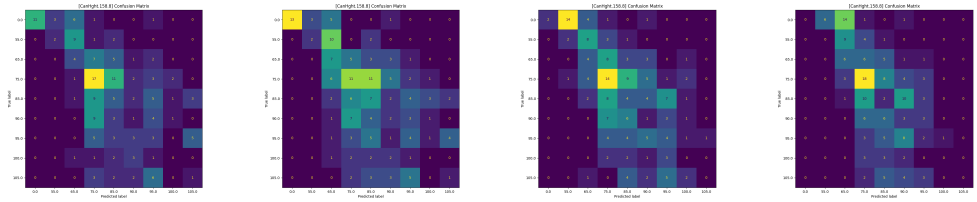


(e) NuSvr (f) LinearSVR (g) FFNN (h) CVNN

Figure D.3: Confusion Matrices for the canopy height on day 119.

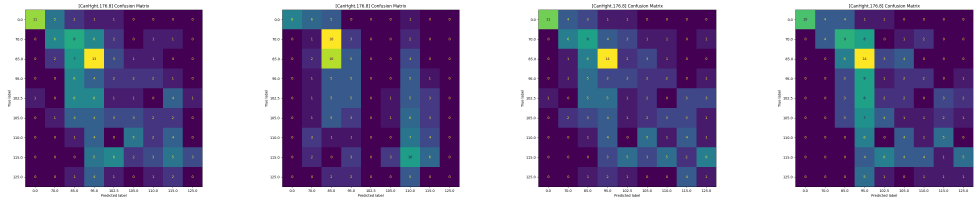


(a) Ridge Regression (b) Decision Tree (c) LassoLarsIC (d) SVR

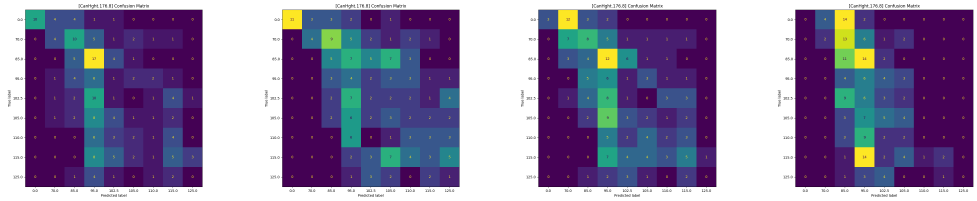


(e) NuSvr (f) LinearSVR (g) FFNN (h) CVNN

Figure D.4: Confusion Matrices for the canopy height on day 158.

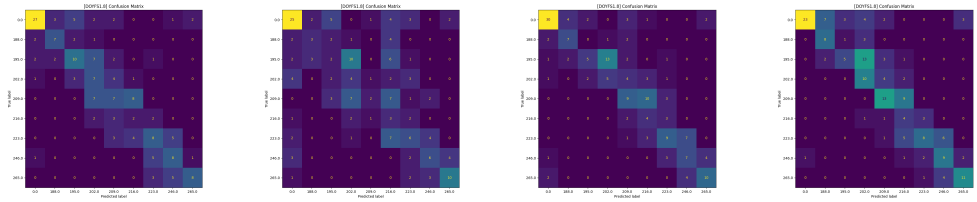


(a) Ridge Regression (b) Decision Tree (c) LassoLarsIC (d) SVR

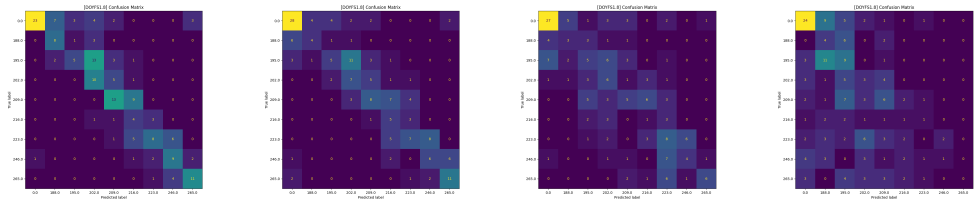


(e) NuSvr (f) LinearSVR (g) FFNN (h) CVNN

Figure D.5: Confusion Matrices for the canopy height on day 176.

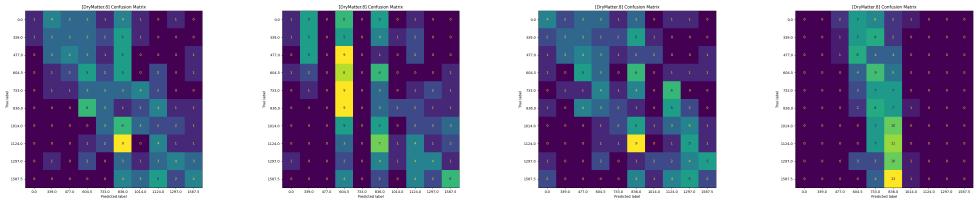


(a) Ridge Regression (b) Decision Tree (c) LassoLarsIC (d) SVR

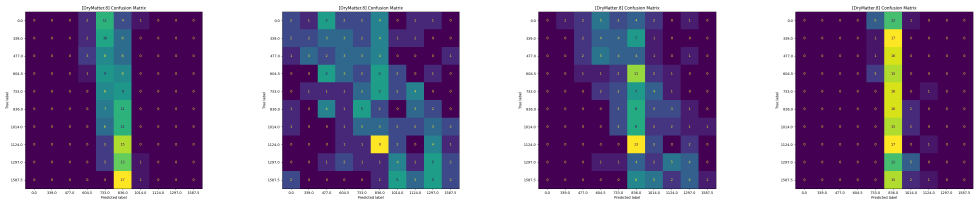


(e) NuSvr (f) LinearSVR (g) FFNN (h) CVNN

Figure D.6: Confusion Matrices for the day of the year that flowering stage one was observed.

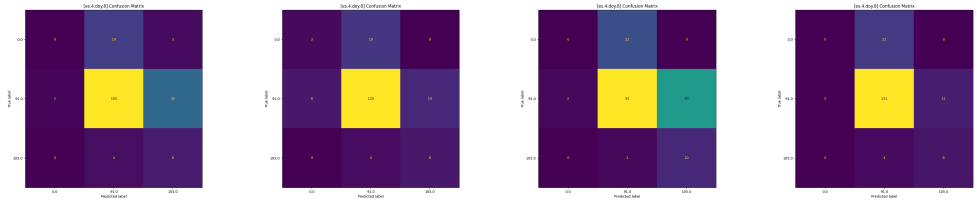


(a) Ridge Regression (b) Decision Tree (c) LassoLarsIC (d) SVR

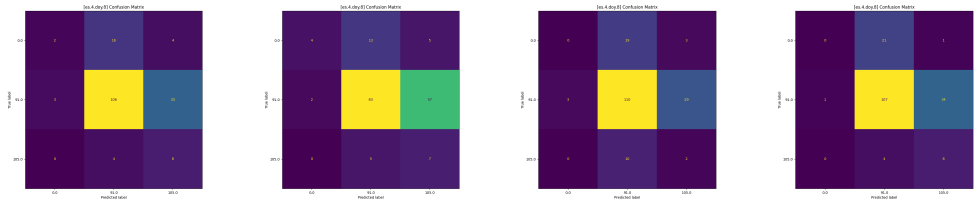


(e) NuSvr (f) LinearSVR (g) FFNN (h) CVNN

Figure D.7: Confusion Matrices for the DryMatter trait.

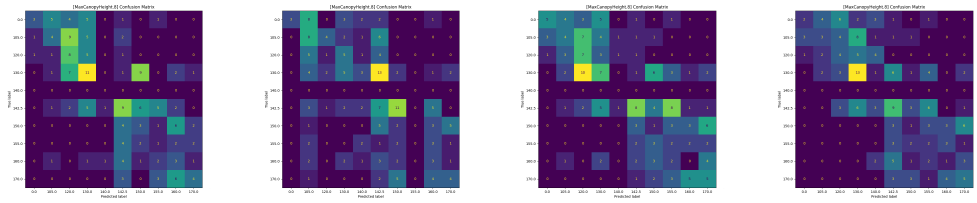


(a) Ridge Regression (b) Decision Tree (c) LassoLarsIC (d) SVR

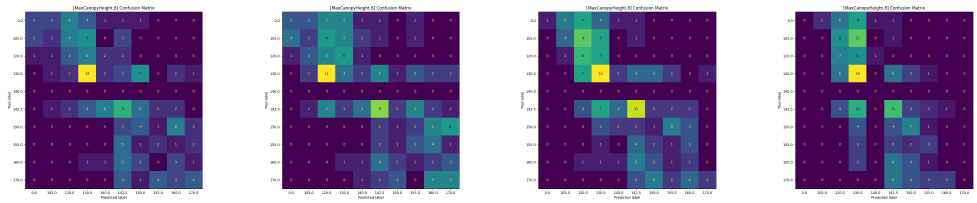


(e) NuSvr (f) LinearSVR (g) FFNN (h) CVNN

Figure D.8: Confusion Matrices for the day of the year emergence stage four was observed.

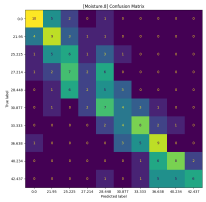


(a) Ridge Regression (b) Decision Tree (c) LassoLarsIC (d) SVR

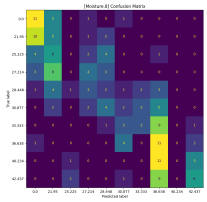


(e) NuSvr (f) LinearSVR (g) FFNN (h) CVNN

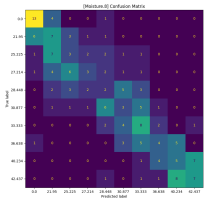
Figure D.9: Confusion Matrices for the maximum canopy height.



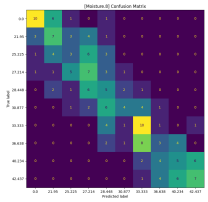
(a) Ridge Regression



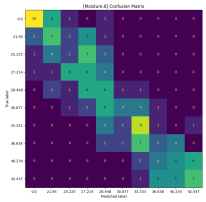
(b) Decision Tree



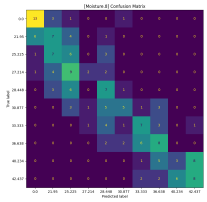
(c) LassoLarsIC



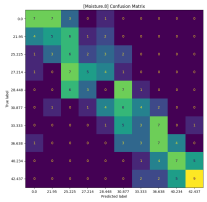
(d) SVR



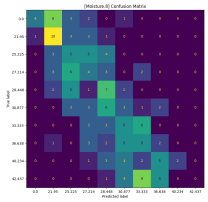
(e) NuSvr



(f) LinearSVR

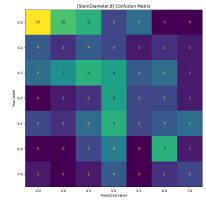


(g) FFNN

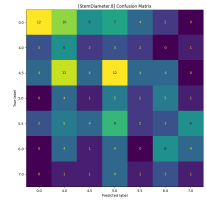


(h) CVNN

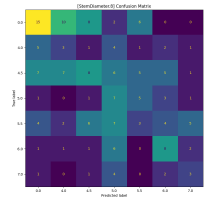
Figure D.10: Confusion Matrices for the Moisture trait.



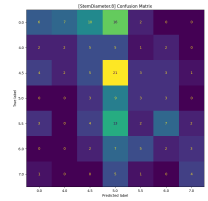
(a) Ridge Regression



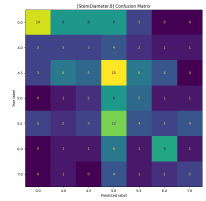
(b) Decision Tree



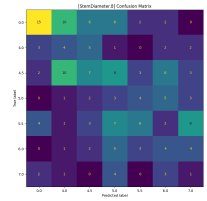
(c) LassoLarsIC



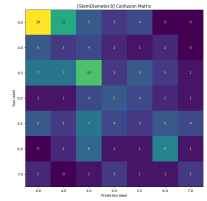
(d) SVR



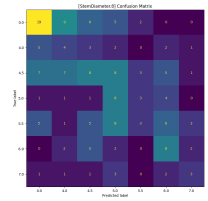
(e) NuSvr



(f) LinearSVR

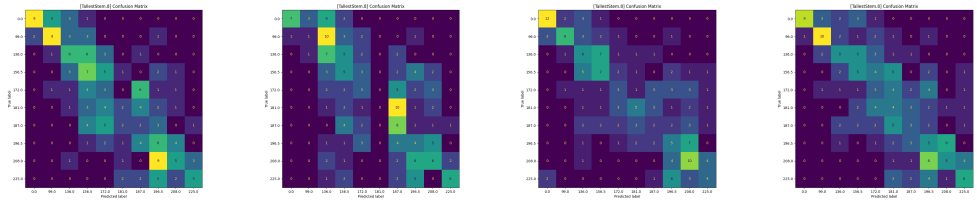


(g) FFNN

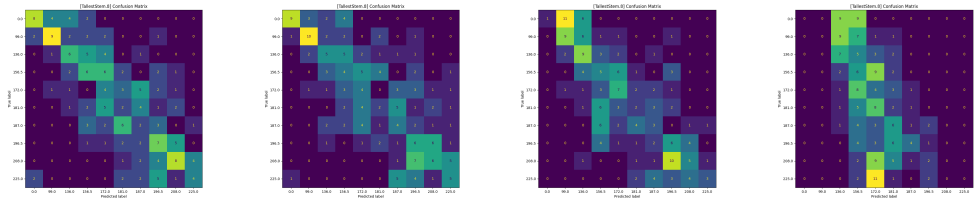


(h) CVNN

Figure D.11: Confusion Matrices for the StemDiameter trait.

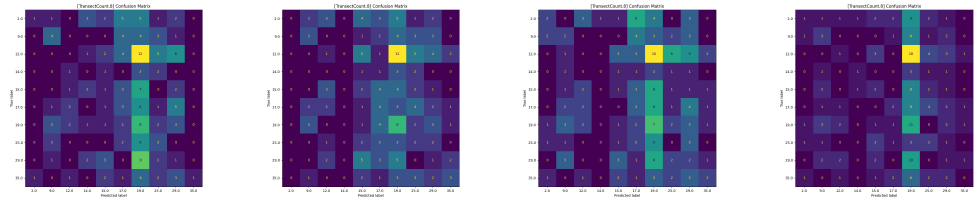


(a) Ridge Regression (b) Decision Tree (c) LassoLarsIC (d) SVR

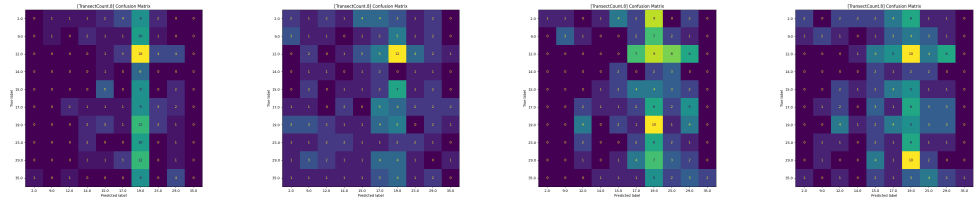


(e) NuSvr (f) LinearSVR (g) FFNN (h) CVNN

Figure D.12: Confusion Matrices for the TallestStem trait.



(a) Ridge Regression (b) Decision Tree (c) LassoLarsIC (d) SVR



(e) NuSvr (f) LinearSVR (g) FFNN (h) CVNN

Figure D.13: Confusion Matrices for the TransectCount trait.