Assignment 4: Random Forest Analysis of Tree Size

Grant Booysen 25849646
Email: 25849646@sun.ac.za

Abstract—**The report investigates how random forest performance varies with decision-tree depth, the number of randomly selected features per split, and ensemble size across three classification datasets of increasing complexity. The goal is to track the performance from underfitting with very shallow trees and few features to overfitting with deep trees and many features, identify points that maximize generalization, and characterize interactions among these hyperparameters. Experiments with (i) maximum depth from minimal to deep trees, (ii) number of features to split on with depth fixed at the best value from (i), and (iii) the balance number of trees with depth were conducted. A final experiment creates ensembles that mix under- and overfitting trees. Models are evaluated with train/test accuracy and 5-fold cross-validation. Findings are consistent: increased depth improves test accuracy up to a point, after which overfitting degrades or stagnates performance and the optimal depth increases with problem complexity. With depth fixed, increased number of features per split improves performance up to a plateau where medium complexity favors moderate feature subsets, while complex data benefits from larger subsets. The inclusion of more trees in the ensemble improves stability and accuracy up to a plateau, with gains moderated by tree depth. Bagging with only overfitted trees proved to have either the best stability, cross validation accuracy or test accuracy across all three datasets.**

Overall, the best configurations balance depth, feature subsampling, and ensemble size where larger values are generally preferred as task complexity rises.

Index Terms—**Random Forest, Ensemble learning, Bagging, Decision Trees**

I. Introduction

Random forests are widely used for classification due to strong generalization, robustness, and modest tuning requirements. The random forest model's performance is controlled by key hyperparameters that trade off bias and variance such as: maximum decision-tree depth, the number of randomly selected features per split, and ensemble size. The report aims to provide understanding for how these controls move the model to better performance. The idea is to identify configurations that generalize across problem complexities.

The goal of this study is to characterize how tree depth and feature subsampling influence random forest performance, and to identify configurations that maximize test accuracy and cross-validated performance across datasets of increasing difficulty. The analysis is achieved by: (i) exploring maximum depth from minimal to deep trees at fixed ensemble size, number of random features per split, and bagging; (ii) fixing depth at the best value from (i) and exploring the optimal number of features to split on per node; (iii) exploring depth and ensemble-size interactions; and (iv) composing ensembles that intentionally mix underfitted and overfitted trees. Evaluation uses train/test accuracy and 5-fold cross-validation to better assess generalization and stability

The report aims to provide a reproducible behaviour for the bias-variance trade-off for random forests, and to extract generalizations for hyperparameter selection that adapts to dataset complexity. The design follows controlled experiments to reveal how each hyperparameter influences performance, and how they interact. The analysis spans three synthetic datasets of increasing complexity to assess how optimal configurations shift with task difficulty.

The main observations are consistent with the bias-variance expectations: - Increasing the depth of the trees in the ensemble improves test accuracy up to a point, after which gains stagnate or degrade; the optimal depth increases with complexity, for example, approximately 8, 15, and 20 on the simple, medium, and complex datasets respectively. - With depth fixed, increasing the number of features to split on improves performance then plateaus; moderate subsets suit medium complexity, while larger subsets benefit complex data. - Adding trees improves stability and accuracy up to a plateau, where the number of trees needed to reach the plateau is dependent on the depth of the trees. - Mixed-depth and overfit-only ensembles: on the simple dataset, majority underfitted trees achieved the highest test accuracy of 0.967 while only overfitted attained the best cross-validated score of 0.966; on the medium dataset, a balanced mix yielded the highest test accuracy of 0.683 while only overfitted achieved the best cross-validated score of 0.691; on the complex dataset, only overfitted trees achieved the highest test accuracy of 0.582 and the most stable cross-validation (CV), with majority overfitted marginally higher CV mean of 0.548.

The remainder of the report is organized as follows: Section 2 provides background; Section 3 details the methodology and implementation; Section 4 describes the empirical procedure; Section 5 presents results and discussion; Section 6 concludes with implications and future directions.

II. Background

The background section discusses the fundamental concepts and algorithms used in the report, including decision trees, ensemble learning, and random forests.

A. Decision Trees

The decision tree is a supervised learning algorithm used for classification and regression tasks. The decision tree works by recursively splitting the data into subsets based on feature values, creating a tree-like model of decisions. Each internal node represents a condition on a feature, each branch represents the outcome of the test, and each leaf node represents a class label in classification or a continuous value in regression. Decision trees are easy to interpret and visualize, but they can be prone to overfitting, especially with deep trees.

B. Ensemble Learning: Bagging

Bagging, or Bootstrap Aggregating, is an ensemble learning technique that aims to improve the stability and accuracy of machine learning algorithms. The algorithm works by training multiple models on different subsets of the training data, which are created by randomly sampling with replacement known as bootstrapping. The final prediction is made by aggregating the predictions of all individual models, typically by averaging for regression or majority voting for classification. Bagging helps reduce variance and combat overfitting, making the approach particularly effective for high-variance models like decision trees.

C. Random Forests

Random forests are an ensemble learning method that combines multiple decision trees to improve predictive performance and control overfitting. The key idea is to leverage the diversity of individual trees by training them on different subsets of the data and features. Traditionally, random forests use bagging to create diverse training sets for each tree. In random forests, it is expected that each tree should be grown to the maximum depth and overfit. The high variance of each tree is then averaged out by the ensemble, leading to a final model that generalizes well.

Algorithm 1 Random Forest (high-level pseudocode)

```
Require: Training data $(X, y)$, n_estimators, max_depth,
    max_features, bootstrap size
1: for each tree $t = 1 \ldots$ n_estimators do
2:     Draw a bootstrap sample from $(X, y)$
3:     Grow a decision tree with depth limit max_depth
4:     At each node, consider max_features randomly selected
       features
5: end for
6: For prediction, average class probabilities (or majority
   vote) across trees
```

III. Implementation

The implementation section describes the datasets, data preparation, and the random forest model used in the experiments.

A. Data preparation

Three synthetic, tabular classification datasets of increasing complexity were programmatically generated using `make_classification` from `sklearn.datasets` [Pedregosa et al., 2011] to enable controlled bias-variance studies and avoid confounds from missing values or categorical encoding. A single `random_state` ensured reproducibility for generation and splitting.

Datasets (generation intent and parameters):

- Simple: binary classification, low noise, no overlap between classes, `n_samples=500`, `n_features=10` where eight are informative and two are redundant, `class_sep=2.0`, `flip_y=0.0`. Designed to saturate quickly with shallow trees and show limited overfitting.
- Medium: three classes, moderate overlap, mild noise, `n_samples=2000`, `n_features=25` where 18 are informative and five are redundant, `class_sep=1.0`, `flip_y=0.2`. Targets visible bias-variance trade-offs at medium depths.
- Complex: four classes, many features, higher noise, `n_samples=6000`, `n_features=50` where 35 are informative and ten are redundant, `class_sep=0.7`, `flip_y=0.3`. Encourages deeper trees and progressive overfitting.

Train/test split: A stratified split with `test_size=0.30` and fixed `random_state` preserves class proportions across splits and guarantees identical partitions across runs.

Preprocessing: No scaling or normalization is applied as decision trees are insensitive to monotonic feature transforms. No imputation is required as the synthetic data contain no missing values. No label encoding is needed as the labels are already integers.

Depth ranges for experimental control: Per-dataset depth grids are predefined to bound sweeps and reflect expected capacity needs:

B. Random forest Model

A custom scikit-learn compatible estimator was implemented by extending `BaseEstimator` and `ClassifierMixin`. Each base learner is a `DecisionTreeClassifier`. The ensemble exposes the hyperparameters: `n_estimators`, `max_depth`, `max_features` (per-split random subspace), `bootstrap`, `criterion`, `min_samples_*`, `max_leaf_nodes`, and `ccp_alpha`. These parameters allow control over tree complexity and randomness. Each individual tree needed to be able to be individually controlled to allow for the mixed underfit/overfit experiments.

Training uses standard bagging: for each tree, a bootstrap sample is drawn. A distinct, reproducible `random_state` is generated per tree from a generator seeded by the estimator's `random_state`. Each tree `max_features` parameter enforces feature subsampling at each split, mirroring the random subspace mechanism. Prediction averages class probabilities across trees uses majority vote.

The model has the following characteristics:

- Equal-weight averaging across trees with no out-of-bag estimates or class weighting by default.
- No post-hoc pruning beyond optional `ccp_alpha` as overfitting is intentionally permitted to study depth effects.
- Deterministic behavior controlled by `random_state`.

C. Cross-validation procedure

Across all experiments, performance for each hyperparameter setting was estimated with K-fold stratified cross-validation on the training split only using `StratifiedKFold` from `sklearn.model_selection` [Pedregosa et al., 2011], reporting the mean and standard deviation of accuracy. Concretely, for the tree-depth, feature-selection, and depth by trees grids, stratified folds for classification were used to compute cross-validated accuracy scores, and record `cv_mean` and `cv_std`. The held-out test split is evaluated separately to visualize train and test gaps and provide a single final generalization number, but not to select hyperparameters. Where an optimal depth is required to center configurations, the depth is chosen by maximizing the CV mean from the prior depth experiment.

IV. Empirical Procedure

The emperical procedure section details the datasets, performance measures, control parameters, and the four experiments executed to answer the study goals. All experiments are fully reproducible given the seeds and grids below.

A. Benchmarks and Splitting

The three datasets Simple, Medium, and Complex were generated as described in the Data Preparation section. Each dataset was split into a training set (70%) and a held-out test set (30%) using a stratified split to maintain class proportions.

B. Performance Measures and Validation

The primary metric used was accuracy and was reported for train, test, and 5-fold stratified cross-validation on the training split. For each configuration, the following are recorded: train accuracy, test accuracy and CV mean and standard deviation. The train-test gap is also computed to assess overfitting.

C. Implementation and Controls

A global seed of 42 was used. Per-tree seeds are drawn from a generator seeded by this value to ensure determinism. Unless varied explicitly, defaults for the decision trees are:

- Criterion for split quality is the `gini` impurity.
- Minimum samples to split a node is two.
- Minimum samples per leaf is one.
- No pruning.
- Bootstrap sampling.
- Equal weighting across trees.

Dataset-specific depth grids: Each dataset has a predefined grid of maximum depths to explore, reflecting expected capacity needs:

- Simple: $\{1, 2, 3, 4, 5, 6, 8, 10, 12\}$
- Medium: $\{1, 3, 5, 7, 10, 12, 15, 18, 20, 25, 30\}$
- Complex: $\{1, 3, 5, 7, 10, 12, 15, 20, 25, 30, 35, 40\}$

D. Cross-validation for Experiments

All experiments use 5-fold stratified cross-validation on the training split (StratifiedKFold). For each hyperparameter configuration we report the cross-validation mean and standard deviation, and rebuild models or ensembles within each fold when applicable. The held-out test split is evaluated separately to report train and test accuracy and the train-test gap. The same reproducible controls (global random seed, per-tree seeds, stratified splits) apply across all experiments.

E. Experiment 1: Maximum Tree Depth

The objective of Experiment 1 is to isolate the effect of maximum tree depth on generalization and overfitting. To this end, we sweep the maximum depth from very shallow to very deep trees while holding other factors constant: a forest of decision trees is trained and all features are used at each split so that each tree is as strong as possible. The variable is the maximum depth, swept over the dataset-specific grids defined above. Performance reporting includes training accuracy, held-out test accuracy and the train-test gap.

F. Experiment 2: Number of Features per Split

The objective is to quantify the effect of per-split feature subsampling while holding depth near the optimal values found in Experiment 1. The experiment fixes the maximum depth to the CV optimal values (Simple: `max_depth=8`, Medium: `max_depth=15`, Complex: `max_depth=20`), enables bagging with 30, 60, and 90 trees for the Simple, Medium, and Complex datasets respectively to stabilise results, and sweeps the maximum number of features per split over $\{1,2,3,5,7,10,15,20,25,30,35,40,45,50\}$, skipping values that exceed each dataset's dimensionality. Performance reporting includes training accuracy, held-out test accuracy and the train-test gap; cross-validation mean and standard deviation are reported as described above.

G. Experiment 3: Depth vs Number of Trees

The objective is to study the interaction between ensemble size and tree depth. In this experiment the number of features considered per split is fixed to `max_features='sqrt'` to provide a moderate level of randomness and bagging is enabled. Tree depth is swept over the dataset-specific depth grids while the number of trees is varied per dataset. Performance is reported as training accuracy, held-out test accuracy, and the train-test gap for each configuration. Results are visualized as heatmaps of test accuracy on the depth by number of trees grid to quantify how increasing the number of trees reduces variance up to a plateau and how the benefit of additional trees depends on tree depth. Cross-validation mean and standard deviation are reported as described above.

H. Experiment 4: Mixed-Depth Ensembles

This experiment evaluates ensembles that mix underfitting and overfitting trees to probe bias-variance blending. Controls are fixed where the number of features per split is set to the optimal values from Experiment 2, ensemble sizes and per-dataset maximum tree depths use the optima from Experiment 3, and bagging is enabled. Five ensemble compositions are compared:

- all underfitted;
- majority underfitted (75% underfitted, 25% overfitted);
- balanced (50/50);
- majority overfitted (25% underfitted, 75% overfitted);
- all overfitted.

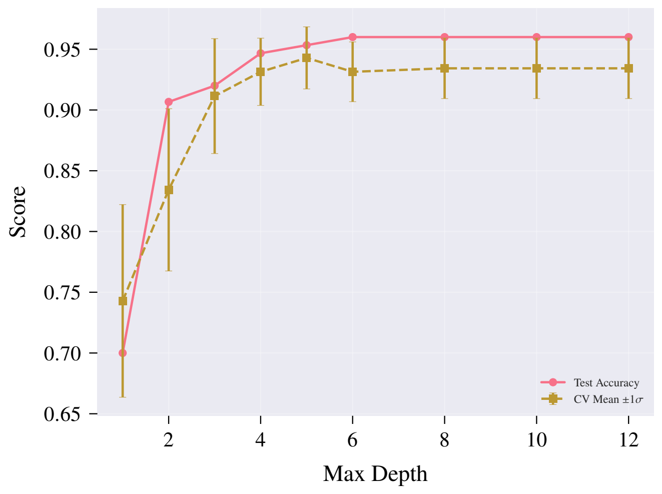
Underfitted depths are sampled from depths below the optimal and overfitted depths from just above the optimal up to the maximum. Performance metrics include training and held-out test accuracy and the train-test gap; cross-validation mean and standard deviation are reported as described above. The goal is to determine whether mixing tree capacities yields improved generalization or stability compared to homogeneous ensembles of the same size.

V. Research Results

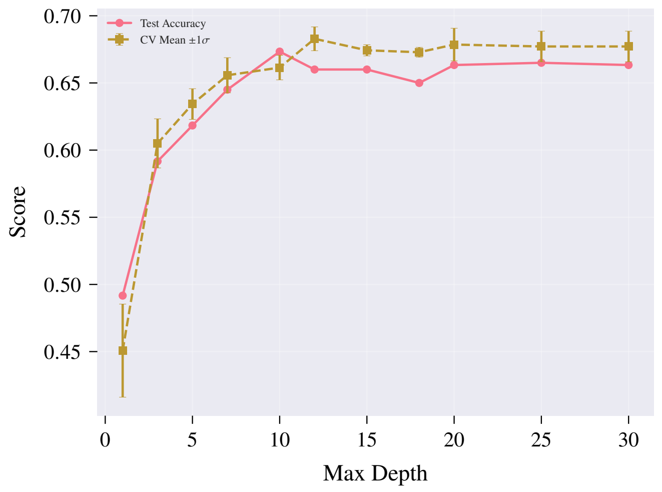
This section reports results for all four experiments. For each configuration the cross-validated accuracy mean and standard deviation over 5 folds on the training split, the held-out test accuracy, and the train test gap are reported.

A. Experiment 1: Maximum Tree Depth

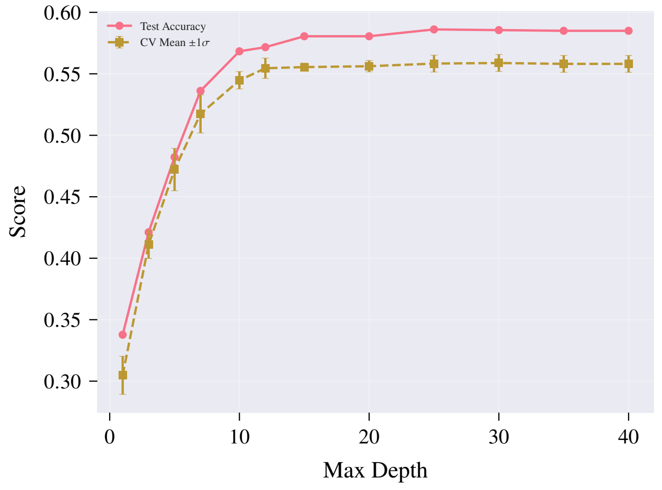
Figures:



(a) Simple: Max depth vs. test and CV accuracy.



(b) Medium: Max depth vs. test and CV accuracy.



(c) Complex: Max depth vs. test and CV accuracy.

Fig. 1: Maximum tree depth vs. test and cross-validated accuracy for the three datasets.

Discussion:

- The results in Figures 1a show the relationship between maximum tree depth and both test and cross-validated accuracy. As expected, very shallow trees in the ensemble underfit the data, which results in low accuracy on both training and test sets. As depth increases, accuracy improves significantly, indicating that the model is capturing more complex patterns in the data. However, beyond a certain depth (e.g., depth 8 for Simple, 15 for Medium, and 20 for Complex), the test accuracy plateaus or slightly decreases while training accuracy continues to rise. This indicates overfitting, where the model learns noise in the training data rather than generalizable patterns. The optimal depths identified align with the complexity of each dataset, with more complex datasets requiring deeper trees to capture their structure.
- These results occur due to the bias-variance trade-off inherent in decision trees. Shallow trees have high bias and low variance, leading to underfitting, while deep trees have low bias and high variance, leading to overfitting. The ensemble averages out some the variance of the overfitted individual trees. However, if there are not enough trees in the ensemble of overfitted trees, the variance does not average out sufficiently, leading to the seen degradation in test performance.
- With more overfitted trees in the ensemble, the variance would average out more effectively, potentially allowing for deeper trees to be used without overfitting.

B. Experiment 2: Number of Features per Split

Quantitative results:

TABLE I: Experiment 2: Accuracy vs. max_features (CV mean ± std and test).

Dataset	MaxFeat	CV Acc	Test Acc	Gap
Simple	1	TBD ± TBD	TBD	TBD
	2	TBD ± TBD	TBD	TBD
	...	...	...	...
Medium	1	TBD ± TBD	TBD	TBD
	...	...	...	...
Complex	1	TBD ± TBD	TBD	TBD
	...	...	...	...

Figures (placeholders):

Fig. 2: Medium: Test and CV accuracy vs. max_features.

Fig. 3: Complex: Train vs. test accuracy across max_features.

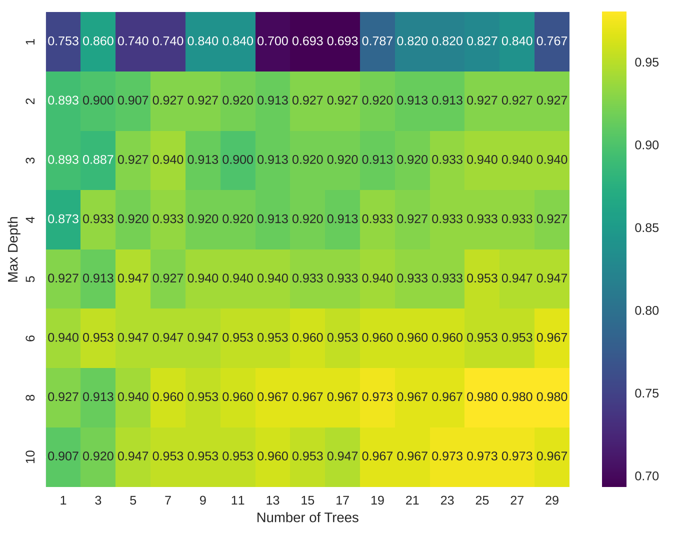
Discussion:

- Expected? Moderate max_features increases tree strength but raises inter-tree correlation; observe plateau.
- Why? Diversity-strength trade-off in bagging; complex data benefits from larger candidate sets.
- Different circumstances? With correlated features, smaller max_features may help; with very high-dimensional sparse data, larger max_features may overfit.

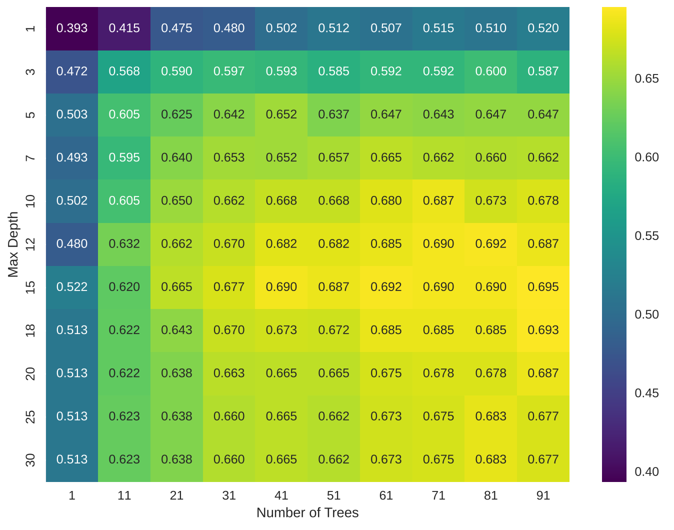
C. Experiment 3: Depth vs Number of Trees

Setup recap: max_features='sqrt', bootstrap=True; grid over depths and n_estimators.

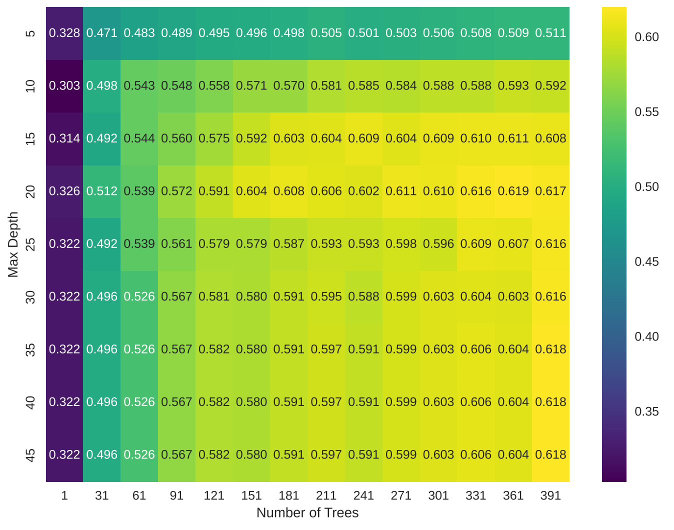
Figures:



(a) Simple (run 1)



(b) Medium



(c) Complex

Fig. 4: Heatmaps of test accuracy over maximum tree depth vs. number of trees for each dataset.

Discussion:

- Expected? Variance reduction with more trees until a plateau; depth-dependent benefits.
- Why? Law of large numbers in bagging; deeper trees have higher variance thus benefit more from larger ensembles.
- Different circumstances? With stronger feature subsampling or more noise, plateau arrives earlier; with larger datasets, plateaus shift upward.

D. *Experiment 4: Mixed-Depth Ensembles*

Setup recap: Five compositions mixing under/overfitted trees; features per split and ensemble sizes from Exp. 2–3 optima; bootstrap=True.

Quantitative results:

TABLE II: Experiment 4: Composition vs. accuracy (CV mean ± std and test).

Dataset	Composition	CV Acc	Test Acc	Gap
Simple	All Underfit	TBD ± TBD	TBD	TBD
	Majority Underfit	TBD ± TBD	TBD	TBD
	Balanced	TBD ± TBD	TBD	TBD
	Majority Overfit	TBD ± TBD	TBD	TBD
	All Overfit	TBD ± TBD	TBD	TBD

Figures (placeholders):

Fig. 5: Medium: Test accuracy by ensemble composition.

Discussion:

- Expected? Depth diversity may improve robustness by blending bias and variance; overfit-only may excel if bagging sufficiently de-correlates trees.
- Why? Weighted averaging of high-variance learners; effect depends on class overlap/noise.
- Different circumstances? With weaker bagging or fewer trees, mixtures may outperform overfit-only; with stronger feature subsampling, underfit proportions may help more.

E. *Ablations and Robustness Checks (Optional)*

- Seed sensitivity: repeat CV with different seeds (TBD) and report variability (TBD).
- Alternate metrics: macro-F1 or balanced accuracy on imbalanced settings (TBD).
- Learning curves: test accuracy vs. train size to confirm data-limited vs. capacity-limited regimes (TBD).

VI. Conclusion

With more computational resources, the model could be made deeper and wider to better learn the relationships in the data. The associated code for all the calculations [1] can be found in the footnote and the references on the last page.

[1]The code for this report is available on GitHub: https://github.com/grantxxcoder/25849646rw441Assignment4.

APPENDIX

LIST OF ACRONYMS

Acronym	**Definition**
CV	Cross-validation

REFERENCES

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