Title Page

Contents

List of Figures

List of Tables

**Abstract**

This project outlines the development of a web-based interactive decision tree builder, allowing complete user configuration for the model building pipeline for both classification and regression. The project creates an easy-to-use, highly configurable model while also producing a layer of abstraction from the technical aspect of model building in machine learning. In turn, the application allows domain experts to use their knowledge to make decisions leading the model development. The web application was built using Python and leveraged the Streamlit framework for web hosting. The sci-kit learn library was used for the decision tree models, and supporting plots, whilst the Dtreeviz library was used for the decision tree visualisation.

Functional, validation and performance testing were carried out on three regression and three classification datasets to assess the training of the models against user expectations. This application successfully demonstrates the ability of improving model interpretability, and explainability through real-time visualisation updates. Users can dynamically tune model hyperparameters, see updated visualisation plots of their configured data and evaluate the performance of their models with warnings. For instances of overfitting or underfitting, the user is introduced to cost complexity pruning, feature selection, GridSearchCV and metrics tools. However, it was found that the scoring metrics defining overfitting and underfitting need to be refined to combat the subjectivity model inaccuracy for different datasets and dataset sizes. Hosting the application as a web application also hinders the performance of large dataset models as load times become significantly high. Further work may introduce Streamlit’s caching functionality and expander widgets to avoid complete page rendering and reduce information overload.

Declaration

Acknowledgements

**Abbreviations**

ECM – Expected Cost of Misclassification

EDA – Exploratory Data Analysis

CSV – Comma Separated Value

PDP – Partial Dependency Plot

MAE – Mean Absolute Error

**Chapter 1 – Introduction**

* Project Overview

Decision trees are used across various sectors, from business market analysis, to healthcare, medicine and finance to make critical decisions (Hogarty, 2022). Since these trees require large amounts of data, evaluating decisions becomes much more complex and consideration is required to ensure decisions being made by these models work for their domain.

One method to increase domain knowledge implementation throughout the model building process is to include expertise from user interaction. By including user interaction, and providing appropriate visualisations with model building support, the user can establish the best model that effectively represents their own data.

Creating a decision tree builder and keeping a human ‘in the loop’ requires the user to gain control of the decisions made by the model, where each change in variable for the tree can be assessed by the user. Giving the user control for each step in creating a decision tree for their dataset will allow the user to make informed decisions for each objective variable in their model. Among others, some of the variables can consider integral hyperparameters of the decision tree, including the maximum depth, and the best split point. This can then be assessed to consider whether simplification of the decision model is needed, or a more intricate approach is required.

This project will evaluate the suitability of a user-configurated decision tree builder application for a range of regression and classification problems. The application will go through extensive functionality and validation testing to assess the usability of the application as an abstraction from the standard low-level machine learning pipeline for use by a non-technical user. The goal of this project is to determine whether a decision tree builder application can be useful within a web-application environment for a variety of different datasets and domains.

**Potential Problems**

* There may be problems with scalability, problems may arise when the dataset is too large and takes up too much memory to perform model building.
* If the dataset is not cleaned properly, any decisions made would still be susceptible to bias.
* Security with confidential data could be an issue with certain datasets, there needs to be a way to ensure datasets that are imported keep data safe when recorded
* Any decisions made will be dependent on the user and their decision. It should be clear that this application and any outcomes made by the application and user are not foolproof and users must take this into account when making critical decisions.
* To provide model building support to users, metrics must be used to help the user identify any overfitting or underfitting. This is a multifactorial approach and can be subjective on the dataset, so caution must be taken when making suggestions.

**Aims**

The aims of this project are to allow full configuration of a classification and regression tree model within a web-based application in Streamlit. Within this application, the aim is for the user to gain full configuration and flexibility over the produced decision tree model, with fully supportive decision tree visualisations and evaluation plots allowing the user to understand the impact of changing one or more variables in the decision tree model. It is hoped that the interactive element of the application will aid the user in implementing fully contextual decisions from the beginning, where they can make informed decisions on what is the best approach for their decision tree model related to their problem statement.

**Objectives**

* Allow the user to modify and visualise the impact of changing objective points, such as max depth, best split point.
* Display appropriate visualisations for model analysis and evaluations
* Allow the user to add in their own dataset easily
* Ensure it can work with a range of datasets
* Work with classification and regression data
* Provide suggestions on the best course of action
* Save configurations and provide history
* Provide an easy-to-use service, perform a user test survey to establish the ease of use of the application

**Report Structure**

• Chapter 2 – Literature Review will cover notable research in decision tree models, overfitting avoidance and current interactive decision tree implementations and the challenges regarding these models.

• Chapter 3 – Design will provide an overview of the choices of tools used and the overall design outcomes along with an initial understanding of the methods chosen for the implementation of the project.

• Chapter 4 – Implementation will cover the steps taken and decisions made in the development of the web-based application.

• Chapter 5 – Evaluation will discuss the testing process used, along with an analysis of limitations found in the data. The results obtained will be validated against the aims of the project.

• Chapter 6 – Conclusion summarises the outcomes of the project with comments from the author. Improvements and suggestions for further research are also addressed here.

**Chapter 2 – Literature Review**

This review aims to cover the current scope of interactive applications for decision tree models, understanding key aspects needed for a successful application and any limitations or challenges in the process. In addition, this section will cover decision trees, their importance in machine learning today, challenges involved with decision tree modelling and the suitability of decision trees within a visual application.

**Decision Trees**

A decision tree model is a treelike hierarchical structure consisting of a root node, internal nodes, branches and leaf nodes, often used in supervised machine learning for predictive modelling (IBM, 2023). The function of a decision tree is to compute a predicted outcome x for a given function (f(x)), completed by representing a set of conditions repeatedly applied from root node to the leaf node. (Breiman et al., 2017; Quinlan,1987).

Decision trees can be used in a number of cases. Song and Lu (2015) review several applications involving decision trees, noting their many uses including variable selection, missing value handling and data manipulation. However, the most common method is for prediction; by using decision tree models to develop prediction algorithms against a target variable, meaningful information can be extracted from the data for the given context to understand the role of which variables play a pivotal role in future predictions. This method was developed by Breiman et al. (2017), who developed the used of Classification and Regression trees (CART) which are widely used in supervised machine learning today. Both trees involve finding a predicted outcome, whether the target variable is categorical (discrete) or numerical (continuous numeric) establishes whether classification or regression trees are used.

Although many supervised machine learning algorithms exist, reviews written by Blockeel et al. (2023) and Shoba and Rangaswamy (2018) state decision trees are often favoured for their interpretability. Any predicted outcome is found through the recursive partitioning of data (i.e. the top-down induction of decision trees) which is both easily computed, understandable and can be graphically visualised in tree form. Expanding on this, little computational power is needed in comparison to other methods, with computation scaling at notation Oand O() for learning and prediction, where n defines the number of rows and m, the number of columns for a given dataset.

Decision trees have had widespread use across machine learning and in industry, with many applications involving the prediction of a medical diagnosis. This process is often found in the classification of well-known datasets such as the PIMA Indians Diabetes (*no date*) and the Breast Cancer Wisconsin (*Diagnostic*) (1993) dataset. The potential applications for predictive decision tree modelling within medicine is vast. Song and Zhang (2014) list several applications, and the impact that classifier and regressor tree models have had within genome and genetic studies. Where models have been used to establish explainability for complex inherited diseases. Other notable applications have previously involved predicted early intervention for neglect and child maltreatment in early stages within Canada (Fallon et al., 2013).

Despite their simplicity and interpretability, decision tree models are prone to overfitting, especially when built for smaller datasets. With the nature of recursive partitioning there requires a need for knowing when to stop splitting. For this, pruning can take place to reduce the level of overfit in the model. As well as decision trees, other predictive modelling algorithms exist such as a random forest. The random forest algorithm is built on decision trees, combining multiple trees in order to reach a singular predictive outcome (Breiman, 2001). Random forest is based on regression trees, building a number of regression trees to average the results. Like singular regression trees, random forest models allow interpretability in estimating the importance of each feature in the model. Moreover, the process of averaging predictions from multiple decision trees can in turn prevent overfitting (*Decision Trees vs Random Forests: Comparing Predictive Power*, no date). As described by Rodriguez-Galliano et al. (2015) in a predictive model for mineral prospectivity, regression trees were better suited in developing explainability for individual features in relation with the target variable. In this case, the regression tree showed more information regarding the effect of different features in relation to mineral deposits.

**Overfitting and Underfitting**

Since decision trees are prone to overfitting, it is important to understand what overfitting is and what can be done to avoid overfitted decision trees. Overfitting is a phenomenon not only associated with decision trees, but within machine learning models in general. So, what is overfitting? Overfitting is where the model interprets the data provided in the training set too closely and cannot generalise well on unseen data. (Bashir et al., 2020; Ying, 2019). Because the model is dependable on the data it is trained on, it struggles find the underlying patterns between the variables and target and performs weakly on test sets (Allamy and Khan, 2014). This can then be recognised when training set accuracy scores are high, and test scores are significantly lower. Underfitting for a model on the other hand, is described for when accuracy scores are considerably low for both training and test scores and little relationship has been identified for the variable and target in the training set for generalisation on new data.

Detecting overfitting within models can be complex and subjective to the dataset. Minimal literature talks about identifying overfitting other than through validation and metrics such as loss and accuracy. In addition, not many methods are found in identifying the size of the difference between the training and test scores statistically as a measure for overfitting. Bashir et al. (2020) created a theoretical approach to identifying overfitting in datasets, finding that their algorithm can estimate when an algorithm will overfit, but only in some cases identify when a model has overfit a dataset. One approach by Khoshgoftaar and Allen (2001) uses misclassification as a measure. Their approach involves finding the Expected Cost of Misclassification (ECM), where the difference between the ECM of a training and test dataset are used to measure overfitting. If this difference is positive, the model is seen to likely be overfitting. Many papers (Gokhale and Lyu, 1997; Allamy and Khan, 2014) describe the methods used to avoid overfitting in the first sense, either through pre- or post-pruning, generalisation cross-validation, or early training stopping.

Blumer et al. (1987) coined the theory of Occam’s Razor, stating that simple models often perform better on unseen data and tend to generalise well. In terms of a decision tree, the theory suggests that there is a trade-off between the complexity of a tree and its ability to generalise. With decision trees, the greater the complexity of the tree the more likely it is to perform more accurately on the dataset. If this training includes all nuances within the dataset, the model could overlook the important patterns within the data for generalisation on the test set. Blumer et al. express that an important balance must be found between the complexity and accuracy to avoid both underfitting and overfitting. Schaffer (1993) disagrees that the Occam’s Razor approach should be a universal truth among models, stating that this method can in fact be treated as a form of bias. Schaffer explains that Occam’s Razor can indeed help manage overfitting by reducing variance, but simplifying models can more than likely introduce bias and underfitting, and model performances should instead be assessed based on the actual data.

**Interactive Model Building**

Many applications for decision tree construction exist to allow for human participation (Ankerst, 1999; Ankerst, Ester and Kriegel., 2000; van den Elsen and van Wijk, 2011). One example is the BaobabView, where Van den Elsen and van Wijk (2011) present an application allowing users to interact, construct and analyse decision trees. It is believed that users often lack technical understanding of decision tree construction but can provide their domain knowledge to improve the process with the support of adequate visualisation and interaction. Ankerst, Ester and Kriegel (2000) further support user interaction and integrating domain expertise in the model building pipeline. Users have the potential to drive more efficient decision tree construction, whilst gaining a deeper understanding of the model tree, for potential improvements in pattern/relationship recognition. However, this approach relies on the use of supporting data and visualisations that are interpretable for a user, where each stage of the model is explainable and can be comprehended by those with less machine learning expertise. Interactive model builders must balance accuracy and interpretability, as well as other factors such as generalisability, robustness and comprehensibility (Gleicher, 2016; Muhlbacher *et al.*, 2018).

Many applications display varying interactions, one such example is by Teoh and Ma (2003), where a user can draw decision boundaries from 2-d projections to automate a decision tree model with these splitting decisions. Ware et al. (2001) are another example of an interactive classification application, where allowing users to generate models themselves acts as a natural way to increase knowledge in model building and often, users were able to create good models through a 2-d visual interface.

Increasing comprehensibility involves creating appropriate visualisations that can provide a good, complete analysis of the data. For decision trees this can become very difficult with large datasets, where important information can become lost within the model’s complexity and become difficult to analyse. Liu and Salvendy (2007) use an icicle plot to represent the structure of the hierarchical tree, as well as display the distribution of samples for each branch. Integrating multiple visualisations to evaluate and analyse a model has been an effective way to provide a more complete analysis of the data. In order to represent the information in the decision tree van den Elzen and van Wijk (2011) provided multiple visualisations in the form of tree-map, streamgraph and a confusion matrix. Maçãs, Campos and Lourenço (2023) state that although tree-maps are the most common usage, displaying large trees in tree-maps requires a large space to effectively see each section. In representing a decision tree forest, they take a different approach in developing a visual tool that provides multiple views incorporating summary views on the random forest, correlation grids and pie charts to aid in giving a complete view of the model.

Before creating an interactive decision tree model, carrying out a review of the literature was important to establish the characteristics behind decision trees and why decision tree models are a good choice for model interpretability. Furthermore, research was needed to provide methods to avoid problems that can occur through modelling such as overfitting and underfitting. Using this information gives some suggestions for incorporating user aids within the application to reduce inaccurate or ill-generalised models. In the research for interactive decision tree models, many applications consider classification problems. Not much insight is given to the power of interactive decision tree modelling for regression models. This paper aims to establish how using the more recent dtreeviz visualisation library could be an effective way of providing user interaction for increased domain knowledge in the model building pipeline.

**Chapter 3 – Design**

3.1. Introduction

This chapter outlines the design objectives for the interactive decision tree builder application, where the application places emphasis on the induction and model analysis rather than complete pipeline implementation, such as data cleaning and exploratory data analysis (EDA). Here, details regarding the technical architecture for complete real-time interaction is provided. Further information regarding the scope for regression and classification decision tree models and the choice of visualisation tools used are also defined.

**3.2. Technical Architecture**

**A computer screen shot of a diagram

Description automatically generated**

Figure 1: Technical Architecture of the Application

Figure 1 above describes the flow in the architecture of the application. The diagram shows how the interaction from the components provided by the host framework (Streamlit) can update values for the parameters for our decision tree model. Further information highlights how updates are rendered within the web application, and how configurations can be saved and uploaded from within the project file structure.

**3.2.1. Web application overview**

One of the challenges that arises when building machine learning models is understanding the patterns between the data, this understanding is often only acknowledged by those with sufficient domain knowledge such as Subject Matter Experts (SMPs) (van den Elsen and van Wijk, 2011).In order to infer domain knowledge from the beginning throughout the model lifecycle, an interactive approach is taken through the development of a web application. Web applications allow real-time monitoring of data, and provide accessibility to users, by removing siloes built by low-level code interaction and instead allowing high level model exploration through a graphical interface. The removal of technical barriers associated with machine learning allows more users, especially those without the technical expertise, to draw conclusions from decision tree models with ease.

For this implementation, Streamlit was chosen to host the decision tree builder as a web application and provide a graphical user interface.

**3.2.2. Streamlit**

Streamlit is an open-source Python framework used for delivering Python scripts as interactive web applications. In this case, the framework provided useful components in integrating interaction with the user through sliders, dropdown features and buttons. In addition, it holds capabilities for displaying matplotlib plots, and seamlessly provides layout options for defining a dashboard through columns.

Other frameworks, such as Solara, provide similar qualities and interaction through ipywidgets components. One of the downsides, however, is the lack of support and compatibility with the dtreeviz library; the library of which is integral to the application for displaying the decision tree visualisations. As Streamlit is open source, a strong community is available for support with troubleshooting.

Moreover, since the dtreeviz library is fundamental in the layout of the dashboard for the user, and interaction is enabled through the framework’s own components, Streamlit is the obvious choice for hosting the web application.

**3.2.3. Dtreeviz library**

A lot of research has been conducted into the effective visualisations of decision trees with comparisons being made to decide which plots provide the best insight into the model. Liu and Salvendy (2007) use an icicle plot to represent the structure of the hierarchical tree, as well as display the distribution of samples for each branch. Numerous options are available for displaying the model’s relationship. Scikit-learn provides the plot\_tree method for displaying information within a tree structure along with node information. This node information consists of the feature name, splitting criteria and the number of samples in the data. Export\_text is another function, less informative and less intuitive than plot\_tree, but provides a textual example for the structure of the resulting decision tree. Of course, with larger models this becomes increasingly complex, becoming harder to interpret and read. Visual graphs for displaying the decision tree are the way forward making the model result interpretable and easier to read. Dtreeviz is a more recent development for the decision tree visualisations. Instead of providing only nodal information, it provides a further in-depth visual analysis. For decision tree classifiers each node displays a histogram of the data at each node, with leaves being depicted as pie charts. This visualisation makes it apparent to the user the proportions of which each class is found at each leaf node. For regression this is depicted through a feature space displaying the decision boundaries for each node, and a regression graph with nodal information at each leaf. The library’s ability to combine powerful graphical information within the tree structure further enhances the user’s interpretation of the model.

**3.2.4. Scikit-learn**

Scikit-learn is a fundamental supervised machine learning library that is both free and open-source. It is used for providing sufficient tools in predictive data analysis. In addition to model building, Scikit learn contributes to the ease of preprocessing data through data normalisation with features such as scaling and categorical encoding. Primarily, in this application the library’s abilities in providing an in-built DecisionTreeRegressor and DecisionTreeClassifier models will be used to create the two sub-applications. By using this library, the pipeline for model induction is streamlined and calculating cross-validated scores is made easier.

**3.2.5. File structure**

The file structure is imperative for the function of the application’s navigation and save/upload features. Through Streamlit’s capabilities, navigating multiple pages in the application is made possible through the defining of a hierarchy in the project folder. In this case, the folder ‘pages’ can be used to house both the classification and regression model scripts so users can easily traverse multiple pages through a sidebar in the end application. Including both model scripts in a singular entity increases the end user’s experience by making the application more intuitive, as this removes any need for loading separate applications or expanding functionality through a button interaction. Defining a ‘saves’ folder in the project file structure means user configurations can be saved within a designated folder, and files can be easily found when users decide to upload their saved configurations.

**3.3 User Interface Design**

Building an effective interactive decision tree application requires ensuring a human is in the loop throughout the lifecycle. Attention must be paid to the user experience and user interaction techniques. How will the user change and tune the decision tree? How can they improve the scores? If no information is displayed to provide hints to the user and they have little technical understanding required for the pipeline, how will they understand whether their model is performing as it should?

One of the main aims for the application is to derive more explainable decision tree models, from which the user can identify patterns and assess how the model reaches certain conclusions. Prioritising dynamic visualisations where user’s inputs update the application in real-time, is one way in which the user can analyse their own inputs within the system. Using the dtreeviz library to visualise the decision tree, not only represents the tree structure and nodal information, but provides visualisation plots for each node along the way. Real-time updates to the model accuracy scores further supports the visualisation in explaining what a good model should look like.

Users can tune hyperparameters in the decision tree induction themselves, through Streamlit widgets such as sliders and dropdowns to adjust parameters at model fitting. The DecisionTreeRegressor and DecisionTreeClassifier models have tunable hyperparameters, most notably the *max\_depth,* *min\_samples\_split* and *min\_samples\_leaf*. *Max\_depth* is the parameter most likely to establish any overfitting or underfitting within the model, as how many levels the tree can take determines how many samples are left at each leaf. The number of samples for each split and each leaf can also determine whether a model will overfit (*Machine Learning*, no date), this can also depend on other factors like dataset size. Other parameters to consider is the criterion, and split type. Once each parameter is changed or updated in the Streamlit components, new values are fitted to the model with new parameters. Each change is then dynamically rendered and updated in the web application.

Multiple methods can then be used to warn the user on overfitting or underfitting a model. Hyperparameter warnings, score warnings and plots can be used together to help identify multiple areas in which a fitting type can be determined. Most common methods include comparing the accuracy of the training and test metrics (Ying, 2019), this can then be visualised for easier understanding of the underlying data to determine fit.

To improve the user’s experience and help identify under/overfitting, hints can be provided to the user whilst tuning hyperparameters, and when analysing accuracy scores. However, one singular score does not define whether a model is overfitting, and in this case, visualisations expedite the ability to consolidate the model’s overall performance.

**Typical User Journey**

Once the user has opened the web application, they can navigate between the classifier and regressor model from the sidebar. Intrinsic features are the same in both. For one, the user uploads a cleaned CSV file, and selects a relevant feature for the target column.

Exploratory data analysis for this application is very minimal, the process for cleaning a dataset and exploring relationships in the data are extensive and are dependent on the type of dataset. Instead, this application focuses on the model induction and evaluation phase in an interactive manner. Users can then change the configuration data for the decision tree fitting using either sliders, or dropdowns. The decision tree, on loading, renders a tree with default parameters where users can alter each parameter and see how each change affects the structure of the tree, and their resultant scores. The user can then see and compile cross validation scores to a k-fold of their choosing, to establish the mean and standard deviation scores across the folds. Feature selection with training based on the selected features is also enabled in the model. The user can also carry out a Grid Search CV, to find the best parameters for a decision tree model.

Scoring types for the two models are different, where default options will include a ‘*balanced\_accuracy’* scoring type for the classifier, and a ‘*mean\_absolute\_error’* score for the regressor. ‘*Balanced\_accuracy’* scoring works well against imbalanced classes within datasets and is easy to interpret (scores closer to 1 are of higher accuracy) (Scikit-learn, no date).The mean absolute error is used as the default metric for the regressor for similar reasons: handling imbalanced data, is less sensitive to outliers and is easy to interpret as a measure of error **(**Schneider and Xhafa, 2022**).**

Model analysis and evaluation methods are where the classifier and regression model truly differ. In the next section, more detail is provided into the layout of the applications, and the reason for the choices of each model evaluation plot.

**Model Applications**

Mock designs for the classifier and the regressor applications are shown in figures 2 and 3. Both applications provide a 3-column layout, displayed as a dashboard. By creating a dashboard most elements available in the application are visible at first glance, and by displaying the decision tree structure in middle and at the top any changes are seen easily by the user without the need for constant scrolling.

Differences between the classifier and regressor models are seen underneath the decision tree, where model evaluation takes place. The next sections will expand for the reasoning for these model evaluation differences.

A screenshot of a computer

Description automatically generated

Figure 2: Design mock-up for Classification Model Builder

A screenshot of a computer

Description automatically generated

Figure 3: Design mock-up for Regression Model Builder

**Classification**

**Upload a CSV file:** The functionalities for both the regression and classifier models are the same. If the dataset is under 200MB, the user can upload any CSV file.

**Target Column**: The target column will only display categorical columns, if no categorical columns are found and the feature with classes is numerical then numerical features are added to the dropdown.

**Data Analysis**: For EDA, the user will be displayed with a scatterplot analysis. With this the user has some access for identifying the relationship between the rest of the numerical features. Since EDA is minimal, this is currently the only graph that is used to explore feature relationships.

**Max\_depth slider:** The values for the sliders are the same for both applications. The user is given the range between 1 and 10 for *max\_depth*. This number is capped at 10 to minimise the load times for the decision tree and reduce the complexity of the overall tree structure for interpretability. Although this is dependent on dataset size, reducing the *max\_depth* to 10 will enhance usability for an interactive web-based application. A warning is displayed to low *max\_depth* levels to hint to the user the idea of underfitting of the model. The same is done for high *max\_depth* levels to hint at overfitting.

**Min\_samples\_split slider:** The values for the sliders are the same for both applications. The user is given the range 1-100. This gives the user enough flexibility to experiment with the number for the parameter to establish its effects. A warning is displayed to low min sample values to hint to the user the idea of underfitting within the model. The same is done for high *min\_samples\_split* levels to hint at overfitting.

**Min\_samples\_leaf slider:** The values for the sliders are the same for both applications. The user is given the range 1-50. This gives the user enough flexibility to experiment with the number for the parameter to establish its effects. A warning is displayed to low min sample values to hint to the user the idea of underfitting within the model. The same is done for high *min\_samples\_leaf* levels to hint at overfitting.

**Test\_size slider:** The values for the sliders are the same for both applications. The user is given the range 0.05-0.95. By providing the ability to create a larger *test\_split* size, the user can understand the effects of the split on the overall model’s performance and scores. Similar warnings are given to the user at low/high test split sizes to reduce the chances of overfitting.

**Criterion dropdown**: Options for the criterion hyperparameter differ. For Classification the options can be ‘*gini*’, ‘*entropy*’ or ‘*log\_loss’*.

**Split dropdown**: Options for the split hyperparameter are the same for both Regression and Classification: ‘*best*’ or ‘*random’*.

**Feature Selection:** The feature selection process is the same for both classification and regression. A feature selection plot is shown, to help the user identify the optimal number of features to use within the model, and its effect on the mean scores. The user can use the slider to decide the number of features to select, with an RFE algorithm used to identify the top-most important.

**Scores**: As stated before, these scores will be given as balanced accuracy scores for classification. Where the training score, test score and a generalisation score (the difference between the two) are given. This generalisation score can be used to establish a threshold to identify whether the model is overfitting.

**Confusion Matrix:** To visualise the prediction error margin for each of the classes, a confusion matrix is displayed. One can see the level of misclassification between each of the classes.

**Cross Validation for Model Evaluation**: Cross validation works the same as for regression. The user chooses the number of k-folds for cross validation and a scoring type to find the scores for each fold, the mean and the standard deviation. The difference between the classification and regression models is the scoring type, the full list of scoring types suitable for classification are outlined in the scikit DecisionTreeClassifier documentation.

**Feature Importance:** The feature importance plot is a function provided by scikit’s DecisionTree methods. The plot displays the importance of each feature in the model’s overall performance with the current configuration.

**Cost Complexity Pruning:** One of the mainprocesses in decision tree induction is pruning (van den Elsen and van Wijk, 2011). The overall function is the same for both regression and classification, with all the work being done by scikit’s DecisionTree method. By applying the same parameters in the end decision tree, the user can see the if any pruning has occurred, and if there are any changes to the overall scores.

**GridSearchCV:** The function of the GridSearchCValgorithm is the same for regression and classification. User inputs will add to the parameter grid for both *max\_depth* and *min\_samples\_leaf* features. Users can then perform the grid search to establish the best model from their given parameters, along with another decision tree and scores to determine if this has a better performance.

**Save/Upload:** Save and upload features are the same for both applications. All changed attributes are saved to a config file under ‘saves’, to then be uploaded by the user.

**Regression**

This section will explain any differences from the functionality of the Classification application for conciseness.

**Target Column:** Any columns identified as ‘numerical’ in the dataset are added to the target dropdown. The user can then choose from these options to decide the target column.

**Data Analysis**: For EDA, the user will be displayed with a box plot for each of the numerical columns and a phi-k correlation matrix. With the boxplot, the user can evaluate the number of outliers within the data and if the data needs to be cleaned further. The correlation matrix provides a way of identifying the correlation between each of the features.

**Criterion dropdown**: Options for the criterion hyperparameter differ. For Regression the options can be ‘*squared\_error’*, ‘*friedman\_mse’*, ‘*absolute\_error’*, or ‘*poisson’*.

**True vs Predicted Values:** To evaluate the visual relationship between the true vs predicted values, the regressor used a scatterplot to identify the fit of the data. More points plotted closer to the line show lower error scores, and greater accuracy.

**Cross Validation for Model Evaluation**: The difference between the classification and regression models is the scoring type, the full list of scoring types suitable for regression are outlined in the Scikit DecisionTreeRegressor documentation.

**SHAP Analysis:** The SHAP analysis provides a local explanation for the regression model, establishing the impact of each of the features from each prediction. Further local analysis can be seen through the waterfall plot, evaluating the impact on the first instance.

**Partial Dependency Plot:** For further evaluating of the model’s performance a PDP is shown. This is used to visualise how each of the numerical features of the model impact and effect predictions.

**DESIGN REQUIREMENTS**

1. **Both models:**
   1. A user should be able to upload a CSV file to import a dataset in the application, if there is an unsuccessful upload a reasoned warning is given to the user
   2. For a given model, any max depth change in the relevant slider should be consistent with the decision tree visualisation and scores. Relevant warnings of small or high max depth levels should be shown to the user as a model building aid.
   3. For a given model, any change in the ‘split type’ dropdown should be observed in the decision tree visualisation and the scoring metrics. These options should be ‘best’ or ‘random’ in accordance with both the DecisionTreeClassifier and DecisionTreeRegressor documentation
   4. For a given model, any change in the *min\_samples\_split* parameter slider should be observed in the decision tree visualisation and scoring metrics. Relevant warnings should be displayed for low values as a model building aid for the user.
   5. For a given model, any change in the *min\_samples\_leaf* parameter slider should be observed in the decision tree visualisation and scoring metrics. Relevant warnings should be displayed for low values as a model building aid for the user.
   6. For a given model, any change in the ‘test\_size’ parameter slider should be observed in the scoring metrics of the configurated tree. Relevant warnings should be displayed for high- and low-test size values as a model building aid for the user.
   7. For a given model, any change in the configuration data and parameters should cause a change in the decision tree visualisation and/or scoring metrics.
   8. For any given model, the cross-validation feature should update the score list, standard deviation and mean scores to represent the selected k-fold by the user. All scoring types should show valid scoring elements.
   9. For any given model, if the pruned tree found through cost complexity pruning has returned fewer nodes than the original user configurated model this should be represented in the scoring metrics and visualisation.
   10. For any given model, the feature importance plot correctly renders in the application with all available non-target features ranked by their overall importance in the model’s performance
   11. For any given model, the feature selection plot should correctly render in the application. With the top-most important features being ranked in accordance with the DecisionTree’s feature importance plot. A new visualisation depicting the selected features, and corresponding scores should correctly render in the application
   12. For any given model, the user input section for the GridSearchCV should only allow valid inputs into the parameter grid, with appropriate error handling for invalid cases. A new visualisation showing a ‘best model’ tree, and parameters should be correctly displayed, matching the parameters found, allowing the user to compare models.
   13. A user should be able to save any configuration of their data on their desktop using an intuitive button feature.
   14. A user should be able to upload any of their saved configurations using an intuitive button feature.
2. **Classification model:**
   1. For a classification model, the user should be able to pick a categorical column for the target using a dropdown, or a numerical target that is inherently categorical if no categorical columns are found.
   2. For a classification model, the user can complete minimal EDA using a feature scatterplot analysis. Here, the correct numerical features should be shown for the axes, with the user being able to configure the x and y axes themselves.
   3. For a classification model, any change in the selection of the criterion dropdown should be shown in the decision tree visualisation and scores. These options should include ‘*gini’*, ‘*entropy*’ and ‘*log\_loss’* to match the specification shown in the scikit DecisionTreeClassifier documentation.
   4. For a classification model that is conventionally overfitting (high *max\_depth*, low *min\_samples\_leaf* and *min\_samples\_split* values), a high training score and lower test score is found, with a generalisation score > 0.1. These scores should display a warning of suggested overfitting.
   5. For a classification model that is conventionally underfitting (low *max\_depth*, higher *min\_samples\_leaf* and *min\_samples\_split* values), training and test scores < 0.7 are found. These scores should display a warning of suggested underfitting.
   6. For a classification model that is conventionally good, training scores > 0.85 and generalisation scores < 0.05 are found. These scores should display an indication of a good model fit.
   7. For a classification model, the configured confusion matrix should display an accurate representation of the test scores for a model that is a good fit, underfitting and overfitting.
3. **Regression Model:**
   1. For a regression model, the user should be able to pick a continuous numerical column for the target using a dropdown
   2. For a regression model, the user can complete a simple EDA using a boxplot and correlation matrix. Both plots should correctly render with correct labels and enlarge for clarity.
   3. For a regression model, any change in the selection of the criterion dropdown should be observed in the decision tree visualisation and scores. These options should include ‘*squared\_error’*, ‘*friedman\_mse’*, ‘*absolute\_error’* and ‘*poisson’*, matching the specification shown in the scikit DecisionTreeRegressor documentation.
   4. For a regression model that is conventionally overfitting (high *max\_depth*, low *min\_samples\_leaf* and *min\_samples\_split* values), a low training error and higher test error is found, with the test error being significantly lower. These scores should display a warning of suggested overfitting.
   5. For a regression model that is conventionally underfitting (low *max\_depth*, higher *min\_samples\_leaf* and *min\_samples\_split* values), training and test errors that are significantly low are found. These scores should display a warning of suggested underfitting.
   6. For a regression model that is conventionally good, training and test errors are shown to be significantly high and have a statistically small difference. These scores should display an indication of a good model fit.
   7. For a regression model, the configured ‘true vs predicted’ scatterplot should render correctly in the application, showing an accurate representation of the training and test errors and supporting any suggestions made by the metrics.
   8. For a regression model, any changes made to the configuration data will update and successfully render the SHAP summary and waterfall plots, with both plots correctly labelled.
   9. For a regression mode, any changes made to the configuration data will update and successfully render the Partial Dependency Plots, with all non-target features being plotted and labelled correctly.

**Conclusion**

Designing an interactive decision tree model must consider a user’s experience. To create an effective interactive application, the user must have full control and be kept in the loop throughout the model lifecycle. As the application is developed to provide a no-code platform for those with little machine learning experience, hints and tips must be provided along the way.

Moreover, prioritising visualisation is integral to increasing model explainability, and interpretability for the user. By consolidating the features needed for a regression and classification model induction pipeline, an effective application for decision tree building can be implemented.

**Chapter 4 – Implementation**

The chapter covers the steps taken to implement the interaction decision tree builder application. Based on the design discussed in the section prior, this section outlines the development of the user interaction, the dashboard layout design and model induction features for the classification and regression pipelines. Steps taken for the environment configuration using Python and Streamlit are described, along with challenges faced during the process.

**Environment**

The environment for the application is relatively simple. The project was developed in Python 3.x, with scripts being implemented in Visual Studio IDE for project organisation and simplicity. Further project organisation was enabled through Streamlit, which was installed and used for development of the web interface and hosting on localhost. The project folder follows the structure provided from figure 1, to construct a hierarchical structure handling multiple web pages in a single application. Separating the regression and classification model functionalities improves user experience in making clear distinctions between the two, as well as providing better modularity. Furthermore, this establishes the relevant differences in the code between the two, reducing the risk of formatting errors between different dataset formats. Finally, the root folder contains the ‘requirements.txt’ file to manage dependencies required by the application, ensuring consistency across platforms.

**Python Script Template Setup**

This section outlines the structure of the Python script to attain the current interactive application, covering key steps for handling user interaction, uploading CSV and config files, rendering visualisations and implementing error handling. By incorporating these aspects, it is hoped that the web application provides an easy-to-use interface capable of encapsulating low-level intricacies often required for machine learning model building.

These next sections will cover in more detail the functionality implemented within the Python scripts to integrate user interaction, using Streamlit to achieve these requirements. Further on, more detail will be provided regarding the implementation for specific functionality needed for both the regression and classification models.

**Layout and UI**

The application takes a dashboard-style layout, designed to focus on intuitive navigation in the application and the reduction of information overload. Streamlit provides functionality for developing front-end design and developing the structure for the layout. This layout is divided into 3 columns, each column specified for specific tasks in the decision tree induction.

Column 1 overlooks the upload of the dataset in CSV format and handles the decision tree configuration – of which the user will specify hyperparameters for fitting. The user here can explore the configuration parameters, take a first look at the data (through either a scatterplot or correlation matrix) or carry out feature selection.

Column 2 is displayed as a larger column to accommodate for the size of the decision tree visualisation. This column is dedicated to rendering all main visualisations for the user configured decision tree. In conjunction with the visualisations, any scores or metrics related to this model evaluation is displayed in this column.

Column 3 is used to conduct a grid search cross validation. Since this is separate from the user configurated decision tree, a new ‘best model’ decision tree is configured. Positioning the grid search elements to the right allows the user to easily make comparisons between the two decision tree structures, and any differences in the parameters to aid in improving their model’s performance.

Buttons for the ‘save’ and ‘upload’ for configurations are displayed at the top of the screen. Positioning the buttons at the top mean the functionality is not hidden for the user, and the placement is consistent across both model pages. Keeping layouts consistent between the two pages eases navigation for users and is intuitive.

**Fundamental Features**

Both the classification and regression models follow the same structure outlined under layout and UI. Here, details for the implementation of fundamental components to the two applications are described.

The primary step in the application is to add a dataset to the application. For this, Streamlit provides a file uploader widget to add the dataset into the model building pipeline. The type is specified as CSV files only as a step for error handling. The uploader has a max size limit of 200MB, so an error message is displayed with Streamlit’s internal error component to indicate the file’s incompatibility.

The save functionality is implemented by saving key information including the dataset and user-controlled variables into a dictionary. All user-controlled values are given default values at the start up until the time of change. Once the save button is clicked, a new configuration file is opened, with all variables written to the new file. This file is then saved under the name of a unique timestamp for easier identification. In terms of data privacy consideration, these files are saved onto the user’s local desktop. Since no public or cloud storage is used any data used in the application remains private. This responsibility is given to the user with respect to their file security maintenance. Moreover, on upload the information is not stored in the application and is reset upon refresh, adding to data privacy preservation.

The upload function uses Streamlit’s file uploader to allow the user to open a .ini file. Config\_parser and StringIO are used to open and decode the information written in the configuration file, where values written are assigned to the values in the current application. Information from the configuration file is uploaded as a string, this string is then converted into the correct relative datatypes to ensure correct variable loading in the application (for example *max\_depth* is casted as an integer on file upload).

The dtreeviz library is used to generate the decision tree visualisation using the library’s in-built viewing functions. The pipeline created including the DecisionTree model is passed to the model function and viewed. However, rendering the visualisation using the inherent view method does not work in Streamlit-based applications. Instead, a function is used to convert the visualisation to an SVG image using repr\_svg to render on the web interface. If errors do arise using the library, and displaying the visualisation an exception is raised, displaying the error information to the user.

Creating interaction elements is vital in giving the user control in the configuration of their decision tree. Two Streamlit widgets are used to update and change variables by the user. Sliders were used for the *max\_depth*, *min\_samples\_split,* and min\_samples\_leaf for the model configuration. These sliders were capped at different ranges to allow flexibility to the user but reduce error margins of erroneous values. For *max\_depth* the user can decide between 1 and 10 depths, for the *min\_samples\_split* the user can choose between 1 and 100 samples per split, or 1-50 for *min\_samples* per leaf. Each change updated by the user in the slider is re-renders the application and the user can see changes in the decision tree structure and scoring. The slider is also used to assign a test split size from 0.05 to 0.95: where the range is defined to reduce the risk of error in the application with erroneous values. For feature selection, the slider is used to select the number of top-most features to include in the model configuration, from range 1 to n where n is the number of features available in the dataset. The slider is also used to allow the user to change the number of k-folds in cross validation from a range of 2 – 10. This slider starts at 2 since cross validation cannot occur with 1-fold, acting as additional error handling.

Dropdowns are used where there is a list of predefined options the user can choose from. Using dropdowns, the user can pick which feature from the dataset will define the target column. For model configuration, dropdowns are used to decide between the criteria parameter and split type. For classification and regression, the criterion type remains different, to ensure the user picks the correct criterion type the options are predefined using scikit-learn’s DecisionTree documentation. The split type is the same for both models and are manually defined as either ‘best’ or ‘random’. Cross validation scoring also differs between the two models, depending on the model type a similar approach is taken where options are predefined for regression/classification scoring types outlined in Scikit learn documentation.

The model evaluation and analysis pipeline for the application includes feature selection, feature importance, cost complexity pruning and a grid search. These methods are handled by scikit learn and function the same for both classification and regression. Feature selection is carried out using Recursive Feature Elimination Cross Validation (RFECV), to identify the topmost performing features in the dataset. This information is plotted using matplotlib and Streamlit’s pyplot renderer, showing the user the mean test scores against the number of features selected. The user then can choose the number of features to select with information provided in this plot. A button is provided allowing the user to retrain the decision tree with the new dataset parameters and compare the final decision tree visualisations and scores to a model utilising the full dataset.

Feature importance and cost complexity pruning are evaluation methods provided by Scikit’s DecisionTreeRegressor and DecisionTreeClassifier classes. Feature\_importance is an in-built function of the DecisionTree class, to display this information visually to the user the data is presented numerically in a dataframe in the application as well as a bar plot showing the ranked importance of each feature in the model’s performance. Cost complexity pruning involves using the cost\_complexity\_pruning\_path provided by the class, this information provides data on the level of impurities at each node as well as the *cpp\_alpha* value – a value that increases when more nodes are pruned. Visualising this data for the user involved a series of matplotlib plots displaying: the total impurity of leaves against the ccp values for the training set, depth and the number of nodes against the *ccp\_alpha*, and the accuracy against the ccp\_alphas for both the test and training sets. These plots provide a greater analysis of the effects of pruning on the end configured tree, supplemented by an additional pruned tree visualisation along with scores to provide a comparison between the user’s original and pruned model configuration.

Creating the GridSearchCV functionality involved allowing the user to input their own parameter grid values for the max\_depth and min\_samples in the model. To do this, there was no widget available in Streamlit to input a list of numbers separately into an array. Since Streamlit is an open\_source framework, many solutions are created by the community and shared. For the input functionality st\_tags (REFERENCE) was used to input singular numbers into a list for the max\_depth and min\_samples array, to then be arranged into the parameter grid. Further error handling was implemented for analysing the inputs as the parameter grid cannot handle negative, float or alpha values. Any non-numerical integer value inputted into the array is identified by python’s isdigit() method and entered into a warning stating these values are not allowed. An as extra measure – these values are not added to the max\_depth or min\_samples arrays. Ther user then uses a Streamlit button to perform the grid search. The best model parameters, decision tree visualisation and final score metrics are evaluated and are shown to allow comparison between other configurated trees.

**Classification Model Implementation**

Defining the target column in the classification model requires options in the target dropdown to only be categorical. Features in the dataset are split into categorical and numerical types, with categorical features being defined as ‘bool’, ‘object’ or ‘categorical’. Alternately, numerical columns are categorised by ‘numerical’ datatypes. From exploratory testing however, some categorical columns are defined as ‘numerical’. For example, the breast-cancer dataset evaluates outcomes as 0 or 1, despite this being a classification problem. This problem could be avoided through thorough data preparation, but to add robustness to the application numerical columns are chosen for the target column options if no categorical features are found. When using categorical data as the target, label\_encoding by scikit learning is used to preprocess the data. This is done as using pure categorical columns in the data caused problems in the decision tree model later in the pipeline, either with displaying or model fitting.

As this is a classification problem, scikit’s DecisionTreeClassifier is used to model the decision tree. Most hyperparameters stay consistent between the two models, except for the criterion dropdown. The criterion for classification has 3 options, ‘*gini’*, ‘*entropy’* and ‘*log\_loss’* as written in the documentation. Scoring metrics for classification are also different, where f1 scoring, accuracy scoring and many more can be used. Scoring is taken by retrieving the prediction results on the training and the test data using a balanced\_accuracy score. Overfitting is often identified when the training score is much larger than the test score (Ying, 2019). To identify how big this difference must be a generalisation score is created which calculates the difference between the two scores. Warnings for overfitting are placed under the scores when the training score is higher than 0.7 and the generalisation score is > 0.1. Underfitting often occurs when scores in training and test scores are low**.** In this case warnings are placed under the scores when both scores show values < 0.7, as scores could suggest underfitting. It is important to note that these warnings are suggestions, as identifying under/overfitting in the data is complex and requires multiple perspectives.

A plot unique to the classification model is the confusion matrix. This uses the ConfusionMatrixDisplay class by sci-kit learn. This represents the accuracy scores visually in terms of the classifications of each class, depicting the number of misclassifications in true vs predicted values.

The classification model implementation can be seen in Appendix X

**Regression Model Implementation**

Unlike the classification model, the target column requires features to be continuous numerical datatypes. Because of this, only features classified as numerical are added to the dropdown list before one-hot encoding for categorical features in the dataset takes place. Preprocessing the data before model fitting is a crucial step that could heavily impact the performance of the regression model. In this case all numerical features are normalised using min\_max scaling and any categorical features in the dataset are one hot encoded to ensure smooth model fitting without errors. Min-max scaling is used to transform numerical features in the range of 0-1, by doing this it ensures the model does not prioritise certain features over others, this can also be useful in visualising graphs, representing them on the same scale. Machine learning models cannot inherently process categorical data, because of this encoding is used to convert these features into numerical ones separated by each class in new columns. If the class is found the value is given a 1, this method ensures there are no incorrect assumptions if it were to introduce ordinality in the model relationship. This process can be a disadvantage when dealing with large datasets or many classes, as the number of features increases by each separate category.

The model is created using scikit’s DecisionTreeRegressor. The class follows the same structure as the classifier; however, the criterion options differ; options measuring the quality of the split are the ‘*squared\_error’*, ‘*friedman\_mse’*, ‘*absolute\_error’* and ‘*poisson*’.Scoring metrics for cross validation are also different, the choices for the regression model is minimal in comparison to classification with only four options: *neg\_mean\_absolute\_error*, *neg\_mean\_squared\_erro*r*, explained\_variance* and *r2*. The performance metric for the user-configured decision tree model uses the mean absolute error (MAE). The MAE score is shown below and acts as a measure of the average difference between true and predicted values, it is an explainable metric of the model’s performance.

(4.1)

= mean absolute error

= prediction

= true value

= total number of data points

Helping the user identify under or overfitting using the scores is more complex with regression scores. A new approach is taken to help identify high errors which can lead to underfitting, and differences in low test and training errors to establish overfitting. A statistical threshold is created by finding the mean and standard deviation through a 5-fold *neg\_mean\_absolute\_*error cross validation. With this, a more robust value for the mean score can be found for the training error and a baseline threshold can be calculated. This threshold is set as the mean + 1.5\*the standard deviation. Overfitting is found if the training error is lower than 0.1 and the test error is greater than the given threshold. Underfitting is seen if both the values are greater than the threshold values. These warnings are displayed as a suggestion, as detection for under and overfitting is complex and is dependent on the dataset.

Three extra plots are created for the evaluation of the regression model. Instead of a confusion matrix as found in classification models, a scatterplot visualising the relationship between the true and predicted values are displayed for both the test and training sets. Creating both allows the user to compare both graphs and supplements the warning to help the user understand the model’s performance and fitting. A SHAP analysis is also displayed as a total summary of the predictions in the model and a local evaluation for the first instance in the dataset. The SHAP summary plot compliments the feature importance plot, providing more information on the SHAP values – displaying how the local SHAP values for each feature are impacted in the model’s overall output. The waterfall plot shown for the first instance expresses how the SHAP values differ for each feature for a local, singular prediction providing more model explainability for the user. With Streamlit, the SHAP library at times struggles to render the plots. To combat this a try catch is added to display the warning to the user and carry on the rendering of other elements in the case of failure. The Partial Dependency Plot is an added plot created through the matplotlib library and sci-kit learn to visualise the global view of each feature on the model’s performance. This establishes the relationship with each features linear relationship with the target feature for each prediction, further adding to the model’s performance explainability.

The final regression model implementation can be seen in Appendix B.

All testing and application files can be found in the GitHub link in Appendix C..

**Challenges and Setbacks**

One of the main setbacks to the implementation of the decision tree builder was found through exploratory testing. In changing the *max\_depth* and configuration values and changing elements for each component in the application it was found that at times some libraries struggle to render all plots. This often happened with the SHAP and Dtreeviz library. Through extensive testing with different datasets and gaining understanding of the data that was input to the models it was found that these libraries would work on and off with the same configuration values. It was found the Streamlit is not always compatible with these libraries and these libraries can be seen as temperamental. Each error that did occur was found in the libraryand was not to do with the application’s code. As a web application is needed to be robust for the user this at times causes major setbacks with rendering all information. To help counter this, try catches were included in all plots to help rectify any unrenderable elements, allowing the user to still load other components in the application. These errors are then shown to the user to explain why some elements are not showing.

**Conclusion**

This chapter reinstates the steps taken in the implementation for the decision tree builder using scikit learn for model building, Streamlit as a web application framework, and the matplotlib and dtreeviz libraries for visualisation. After initial development and exploratory testing, the application has use for both classification and regression, creating explainable decision tree models with thorough model evaluation techniques along with handling real-time model configuration by the user.

**Chapter 5 – Evaluation**

Evaluating the performance of the final application is completed through further functional, validation and usability testing. The testing takes an extensive approach in understanding the application’s performance against the original design requirements found in **Chapter 3** and any user expectations. Each component of the application was tested against a series of specifications focusing on user input, model configuration, scoring and complete rendering. Using three sample datasets of different sizes and characteristics for each model, a repeated approach was taken to determine the application’s reliability, ease-of-use and correctness. Any configurations modelled were verified to replicate overfitting, underfitting and good models, and were compared against the suggested scoring messages to ensure reliability. Performance metrics were also tested for each dataset, with increases in dataset size and *max\_depth* values for observations under high data loads.

**Testing Datasets**

Three datasets for each model were used to carry out compatibility testing. Being widely used and well-researched by the machine learning community, they provide different characteristics and in turn aid in creating a more robust application. Prior to model building, each dataset was cleaned to represent the user’s expected approach to the application. Further information regarding the characteristics of each dataset, and why these were chosen are explained below.

**Classification:**

The Iris dataset is a small dataset representing a multi-class classification problem. Consisting of 150 samples and 4 features, the dataset shows how the model builder handles simple classification problems, predicting the species of an iris flower with 3 classes: ‘iris’, ‘setosa’ and ‘virginica’.

Wine Quality represents a larger dataset of 4898 instances and 11 features. Like the Iris dataset this represents a multi-class classification problem, with the difference being that the target column ‘quality’ is represented numerically instead of categorically. This dataset assesses how the application will handle numerical class features, along with an increased size.

The Breast Cancer Wisconsin dataset has a greater number of features than the Iris and Wine Quality dataset, with 569 instances and 30 features. This dataset presents a binary classification problem predicting a malignant or benign diagnosis and reflects a possible real-world problem within machine learning.

**Regression:**

The PIMA Indians Diabetes dataset is inherently a classification problem, with a numerical class target ‘outcome’. The file contains information about 768 women and 9 features including a diabetes diagnosis and other predictor variables. By assessing the application’s performance against this dataset, it is possible to see how the model handles non-conforming data.

As well as classification problem, the Wine Quality dataset can also be treated as a regression problem. The ‘quality’ target variable is a continuous numerical value and acts as a standard regression problem that can predict a certain quality score.

The California Housing dataset is a large dataset representative of more complex regression tasks. Containing information regarding the properties of housing information and prices, the dataset enlists 9 features and 20,640 instances and predicts housing prices. Testing with this dataset provides an idea of how well the application handles large datasets within an interactive web-based environment.

**Functionality/Validation Testing Results and Analysis**

A validation table was created to streamline the testing process, with each design outcome being divided into actionable tasks for testing. This was used to compare the expected vs actual outcome of the task. A sample of the scoring functionality being tested for a singular classification test can be seen in table 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **Scores** | | | |
| Action | Expected Outcome | Outcome |
| Change configuration criteria | Scores for training, test and generalisation should change and update | Each change in the parameter configuration causes an update for the training, test and generalisation scores. |
| Configure overfitting model (High max depth, low min samples split, low min samples leaf). Check decision tree and visualisation to check this is accurate – is it really overfitting? | High training score, lower test score. Warning is displayed stating that the scores suggest overfitting | Training score is 0.98, test score is 0.86. Since the generalisation score is higher than expected, the user receives a warning displaying that the scores suggest overfitting in the model. Other instances where the training score is high e.g. 0.96 and the test is 0.88 suggest there could be some element of overfitting. The model displays a message stating the model may need some improving. Could this be seen as overfitting instead? |
| Configure underfitting model (low max depth, high min samples split, high min samples leaf). Check decision tree and visualisation to check this is accurate – is it really underfitting? | Low training and low-test scores. Warning is displayed stating that the scores suggest underfitting | When trying to purposely underfit the model, the scores are still relatively high at 0.7. Instead of displaying an underfitting warning from looking at the scores, it instead suggests that the model may need improving. |
| Configure good model (check parameters through grid search CV). Check decision tree and visualisations to check this is accurate. | High training and high-test score. Displays message suggesting that the scores show the model is a good fit. | High training score: 0.913, high test score: 0.897. A message is displayed underneath the scores suggesting this is a good model fit. This is supported by the confusion matrix; few are misclassified without overfitting. |

Table 1: Classification Model - validation testing for scoring

To simplify any evaluations to be made from the results, a testing matrix with generalised features in tables 2 were produced.

|  |  |  |  |
| --- | --- | --- | --- |
| **Regression Model** | | | |
| **Test Case** | **PIMA Indian Diabetes** | **Wine Quality** | **California Housing** |
| **File Handling and Upload** | Handles uploading and saving as expected. Error is found when config file uploaded – replacing with new CSV file does not refresh. | Handles uploading and saving as expected. Error is found when config file uploaded – replacing with new CSV file does not refresh. | Handles uploading and saving as expected. Error is found when config file uploaded – replacing with new CSV file does not refresh. |
| **Target Column Selection** | The target variable ‘outcome’ for classification inherently numerical. Can still be chosen as the option for regression | Works as expected, ‘quality’ is treated as numerical and can be chosen as target column | Target variable ‘median housing value’ is displayed in dropdown. Works as expected. |
| **EDA Rendering *(Boxplot and Correlation Matrix)*** | Displays as expected | Displays as expected | Displays as expected |
| **Model Parameter Configuration *(max\_depth, min\_samples, split, criterion and test\_size)*** | Changes in configuration data observed and warnings displayed as expected. No change is observed in score with *min\_samples* update. | Changes in configuration data observed and warnings displayed as expected. No change is observed in score with *min\_samples* update. | Changes in configuration data observed and warnings displayed as expected. No change is observed in score with *min\_samples* update. |
| **Feature Selection** | Plots displayed as expected, features are ordered by rank of importance | Plots displayed as expected, features are ordered by rank of importance | Plots displayed as expected, features are ordered by rank of importance |
| **Dtreeviz Visualisation** | All changes in configuration data successfully render updated visualisation | Receive a dtreeviz visualisation error at *max\_depth* 9-10. | Large delay in visualisation for *max\_depth* values larger than 6 |
| **Model Scoring Metrics** | Classification problem – the scores struggle to display good models as expected. No obvious underfitting warnings are displayed when comparing scores to the true vs predicted plots. Generic message is only given to improve model. | Overfitting and good models work as expected, correlating with predicted plots. Underfitting warnings do not display as expected. | Underfitting established with low errors because it passed baseline threshold. Overfitting was established when differences was small between errors although one passed threshold. |
| **Evaluation Plots and Visualisation *(True vs Predicted, Feature Importance, SHAP, Cost Complexity Pruning, PDP)*** | All plots rendered without error, as expected. | All plots rendered for lower *max\_depth* values. For *max\_depth* 9-10 plots do not continue to render as expected. | CCP pruned tree at times takes a very long time to load. All other plots display without error as expected. |
| **Cross Validation** | Cross validation with all k-fold values work. All scoring types display valid results. | All cross-validation inputs work as expected, displaying valid cross val results. | Cross validation with all k-fold values work. All scoring types display valid results. |
| **Grid Search CV** | Correct error handling, Grid search performance runs as expected | Correct error handling, when reached grid search performance runs as expected | When grid search cv does load, this works as expected. |
| **Classification Model** | | | |
| **Test Case** | **Iris** | **Wine Quality** | **Breast Cancer** |
| **File Handling and Upload** | Handles uploading and saving as expected. Error is found when config file uploaded – replacing with new CSV file does not refresh. | Handles uploading and saving as expected. Error is found when config file uploaded – replacing with new CSV file does not refresh. | Handles uploading and saving as expected. Error is found when config file uploaded – replacing with new CSV file does not refresh. |
| **Target Column Selection** | Target column selection for categorical target variable works successfully | Works as expected, model defaults to numerical, ‘quality’ can be chosen as target column | Works as expected, model defaults to numerical, ‘diagnosis’ can be chosen as target column |
| **EDA Rendering *(Feature Scatterplot Analysis)*** | Displays as expected, user can change axes features | Displays as expected, user can change axes features | Displays as expected, user can change numerical axes features |
| **Model Parameter Configuration *(max\_depth, min\_samples, split, criterion and test\_size)*** | Changes in configuration data observed and warnings displayed as expected. No change is observed in score with *min\_samples* update. | Changes in configuration data observed and warnings displayed as expected. No change is observed in score with *min\_samples* update. | Changes in configuration data observed and warnings displayed as expected. No change is observed in score with *min\_samples* update. |
| **Feature Selection** | Plots displayed as expected, features are ordered by rank of importance | Plots displayed as expected, features are ordered by rank of importance | Plots displayed as expected, features are ordered by rank of importance |
| **Dtreeviz Visualisation** | All changes in configuration data successfully render updated visualisation | Receive a dtreeviz visualisation error for max\_depths > 6. | All changes in configuration data successfully render updated visualisation, with some delay |
| **Model Scoring Metrics** | Scoring shows good accuracy scores as expected. Overfitting model difficult to achieve due to size of dataset | Overfitting and underfitting errors show correct messages displayed. Good model with accuracy > 0.8 were not found, correct messaged was displayed in this case. | Subjectvity shown in overfitting scores. A good model fit showed 0.96 and 0.88 for training and test. The difference could possibly be established as overfitting. |
| **Evaluation Plots and Visualisation (Confusion Matrix, Feature Importance, Cost Complexity Pruning)** | All plots rendered without error, as expected. | All plots rendered for lower max\_depth values. For max\_depth 6-10 plots do not continue to render as expected without adequate error handling. | All plots display without error as expected. |
| **Cross Validation** | Cross validation with all k-fold values work. Nan values found with ‘f1’, ‘precision’, ‘recall’ This could be due to class imbalance. T | Cross validation with all k-fold values and scoring types work as expected. | Cross validation with all k-fold values work. All scoring types display valid results. |
| **Grid Search CV** | Correct error handling, Grid search performance runs as expected | Correct error handling, when reached grid search performance runs as expected | When grid search cv does load, this works as expected. |

Table 2: Testing Matrix for Regression and Classification Models

This training matrix provides a high-level overview of the results gained from validation testing for each feature in the application. For both models, one of the main problems that arose with the upload configuration feature. Once this was uploaded, the user could not restart the page as expected by changing the CSV file, as the configurations that were loaded were still saved with previous dataset information. To fix this, error handling will need to be introduced to refresh saved variables in the configuration file on upload of a new CSV file. Another indication of a potential problem during testing was with the *min\_samples\_split* and *min\_samples\_leaf* values. As the values increased or decreased, no change in the scoring metrics were observed. This could be due to the size, or characteristics of the datasets so further testing should be carried out to establish any bugs within the hyperparameters themselves.

For the Wine Quality dataset, it was shown that high *max\_depth* values caused incompatibility with the dtreeviz library for both the classification (figure 6) and regression model (figure 5). More than likely, this was be caused by the incompatibility of the dataset with the library, with the errors being returned as internal library focussed errors. This error is a large bottleneck in the application as the visualisation of the tree structure is an integral part of the model evaluation. To assess the impact of this error, more datasets need to be tested, and extensive research into the library’s scripts should be analysed. If the error arises with many more, this could be a deciding factor on the effectiveness of the dtreeviz library in interactive decision tree building.

For cross-validation scoring with the Iris dataset, it was found that *f1*, *precision* and *recall* scoring types returned nan values for each score. These methods are not intended to be used for multi-class classification scoring, which is why macro and weighted versions of these scoring exist. Due to this, an error handling exception should occur when class counts are greater than two.

During testing, some regression models struggle to determine an underfitting model through the mean absolute error. To identify any under or overfitting seen in the data, any conclusions made from the error scores were compared to and supported using the ‘True vs Predicted’ plot. In some cases (particularly the California Housing data) very low training and test errors were found, indicating an accurate model. Although configurations and *max\_depth* were changed to include low and high values, some values were relatively less accurate. Due to this, using the statistical threshold may not have been the best course of identifying an underfitting model, since all errors are relative to the dataset meaning low error scores for one model could mean less accurate results for others. In this case, other statistical measures should be calculated to help users better identify underfitting and overfitting with scores.

A screenshot of a computer

Description automatically generated

Figure 5: Regression model dtreeviz error

A screenshot of a computer

Description automatically generated

Figure 6: Classification model dtreeviz error

**Performance Testing**

What was apparent during validation testing is that larger datasets such as the California Housing dataset and large *max\_depth* values effect the load times for plots and the decision tree visualisations. Performance testing was used to measure how the size of the decision tree structure and the dataset affects the loading times within the web application. This was used to assess if a web application is the best approach in creating an interactive decision tree builder. From calculating the time taken to completely render all features within the application, it was found that there was a clear correlation between the dataset size, *max\_depth* value and the time it takes to load the application, with larger values and size creating longer loading times.

|  |  |  |  |
| --- | --- | --- | --- |
| **Classification Model – Performance Testing** | | | |
| **Dataset name** | **Load\_times (s): Max\_depth = 1, Min\_samples\_split = 2, Min\_samples\_leaf = 1** | **Load\_times(s):**  **Max\_depth=4**  **Min\_samples\_split = 2,**  **Min\_samples\_leaf = 1** | **Load\_times(s):**  **Max\_depth = 7**  **Min\_samples\_split = 2,**  **Min\_samples\_leaf = 1** |
| **Iris** | 20.759237051010132 | 58.206010818481445 | 62.7831158638000 |
| **Wine Quality** | 5.467482805252075 | 18.824524641036987 | **--- N/A** |
| **Breast Cancer** | 5.28674578666687 | 12.457592010498047 | 14.989876985549927 |

Table 3: Performance Testing table for Classification Model

|  |  |  |  |
| --- | --- | --- | --- |
| **Regression Model – Performance Testing** | | | |
| **Dataset name** | **Load\_times: Max\_depth = 1, Min\_samples\_split = 2, Min\_samples\_leaf = 1** | **Load\_times:**  **Max\_depth=4**  **Min\_samples\_split = 2,**  **Min\_samples\_leaf = 1** | **Load\_times:**  **Max\_depth = 7**  **Min\_samples\_split = 2,**  **Min\_samples\_leaf = 1** |
| **Diabetes** | 28.91278600692749 | 89.90188717842102 | 286.8029680252075 |
| **Wine Quality** | 37.40577006340027 | 113.86705088615417 | 375.6883478164673 |
| **California Housing** | 53.99760103225708 | 147.77776980400085 | 707.8037519454956 |

Table 4: Performance Testing table for Regression Model

Figure 7: Classification Models Performance Testing Graph

Figure 8: Regression Model Performance Testing Graph

The results from the tables 3-4 and figures 7-8 reveal the impact of the max\_depth value and size of the datasets in a model’s performance. The tables and graph show that regression models have a larger loading time than that taken for classification models, this could be due to the number of features involved in model fitting. Having such long loading times for larger datasets also suggests that using model building with this methodology may not be sustainable, or scalable. Further improvements must be made to combat longer loading times with larger models, with proposed corrections involving component level loading instead of complete page rendering. This could be achieved with Streamlit’s expander components to separate the loading of individual features, which additionally could reduce information overload and navigation within the application.

**Conclusion**

This section follows the application’s testing process through extensive functional validation testing and performance testing. Despite showing promise with high level model building, it is apparent that features in the application must be refined for improved usability. Testing for each model must include a larger pool of datasets to help identify bugs either within the libraries or application for added robustness. More research into methodologies used for overfitting and underfitting detection using only scoring metrics must be performed, to ensure accuracy in suggestions made for the user. Complete error handling must be considered with complex user processes regarding user uploads and inputs within cross-validation scoring. Moreover, improvements must be made in the scalability handling of the application, where caching or segmentation can be implemented to aid in reducing loading times for larger datasets or high max\_depth values. By improving the usability aspect of the application, the application shows promise as an interactive decision tree model application capable of providing easy to use, explainable models.

**Chapter 6 - Conclusion**

The goal of this project was to design an interactive decision tree builder for classification and regression problems as a web-based application. This application would allow for complete user configuration and interaction with the final decision tree model to infer their own domain knowledge throughout the model building process. With access to supportive decision tree visualisations, evaluation tools and plots it was hoped that this would aid the user in making informed decisions for their decision tree to make a good model that supports their problem statement.

Through validation testing it was found that besides some error handling integrations that can be easily implemented as next steps, there are critical problems within the visualisations of the decision tree model. The dtreeviz library, at times does not support Streamlit visualisations of the SVG file and is corrupted by high max\_depth values using the Wine Quality dataset. In addition to this, although the support messages provided to the users are suggestions, the subjectivity regarding labelling overfitting and underfitting in the model may need to be refined. The current baseline threshold limitation for regression suggests that the method for defining underfit within the application must be able to handle relative scores for different datasets. With the datasets used, larger datasets tend to provide lower error scores. Any thresholds used should consider a threshold related to dataset size for further accuracy regarding user aid. A possible future outcome from this project could include further research in the detection of overfitting within regression, specifically using the scoring metrics.

Another limitation within the application was found through performance testing. What was found was that the increase in the max\_depth size and dataset size were factors increasing the load times significantly. As this is a web-based application, load times should be reduced for a better user experience. Possible future work could include Streamlit’s caching feature to reduce load times on configuration change, as well as use of the expander widget to avoid whole page rendering and reduce information overload for the user.

Despite the dtreeviz library’s integration problems with Streamlit, there is promise in incorporating user domain knowledge within the model pipeline. The user is given full access to all updatable parameters for the decision tree model and can infer the effects of each feature on the model outcome for both classification and regression models. Users are given the ability to select features, perform a grid search to identify best model parameters, as well as prune their configured dataset to combat overfitting within the data. Although not always suitable for larger datasets, this application could be used for smaller datasets and successfully provides an example of how interactive decision tree builders can be created for regression and classification models as a web-based application. Top of Form

**References**

Allamy, Haider & Khan, Rafiqul Zaman. (2014). Methods to Avoid Over-Fitting and Under-Fitting in Supervised Machine Learning (Comparative Study). 163-172. 10.3850/978-981-09-5247-1\_017.

Bashir, D. *et al.* (2020) ‘An Information-Theoretic Perspective on Overfitting and Underfitting’, *AI 2020: Advances in Artificial Intelligence*, pp. 347–358. Available at: https://doi.org/10.1007/978-3-030-64984-5\_27.

Blockeel, H. *et al.* (2023) ‘Decision trees: from efficient prediction to responsible AI’, *Frontiers in artificial intelligence*, 6. Available at: <https://doi.org/10.3389/frai.2023.1124553>.

Blumer, A. *et al.* (1987) ‘Occam’s Razor’, *Information Processing Letters*, 24(6), pp. 377–380. Available at: <https://doi.org/10.1016/0020-0190(87)90114-1>.

Breiman, L. (2001) *Random Forests*. Available at: https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf.

Breiman, L. *et al.* (2017) *Classification And Regression Trees*. Routledge. Available at: https://doi.org/10.1201/9781315139470.

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Wine Quality [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C56S3T.

*Decision Trees vs Random Forests: Comparing Predictive Power* (no date) *dataheadhunters.com*. Available at: <https://dataheadhunters.com/academy/decision-trees-vs-random-forests-comparing-predictive-power/>.

Fallon, B. *et al.* (2013) ‘Opportunities for prevention and intervention with young children: lessons from the Canadian incidence study of reported child abuse and neglect’, *Child and Adolescent Psychiatry and Mental Health*, 7(1). Available at: <https://doi.org/10.1186/1753-2000-7-4>.

Fisher, R. (1936). Iris [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C56C76.

Gleicher, M. (2016) ‘A Framework for Considering Comprehensibility in Modeling’, *Big Data*, 4(2), pp. 75–88. Available at: https://doi.org/10.1089/big.2016.0007.

Hogarty, S. (2022) *Decision trees: Definition, analysis, and examples*, *Ideas*. Available at: <https://www.wework.com/ideas/professional-development/business-solutions/decision-trees-definition-analysis-and-examples>.

IBM (2023) *What is a Decision Tree | IBM*, *www.ibm.com*. Available at: <https://www.ibm.com/topics/decision-trees>.

Khoshgoftaar, T.M., Allen, E.B. Controlling Overfitting in Classification-Tree

Models of Software Quality. Empirical Software Engineering 6, 59–79 (2001). https://doi.org/10.1023/A:1009803004576

Liu, Y. and Gavriel Salvendy (2007) ‘Interactive Visual Decision Tree Classification’, *Springer eBooks*, pp. 92–105. Available at: <https://doi.org/10.1007/978-3-540-73107-8_11>.

Lyu, M.R., & Gokhale, S.S. (1997). Regression Tree Modeling For The Prediction Of Software Quality.

Maçãs, C., Campos, J.R. and Lourenço, N. (2023) ‘Understanding the Forest: A Visualization Tool to Support Decision Tree Analysis’, *2023 27th International Conference Information Visualisation (IV)* [Preprint]. Available at: <https://doi.org/10.1109/iv60283.2023.00047>.

*Machine Learning* (no date) *Google Developers*. Available at: https://developers.google.com/machine-learning/decision-forests/overfitting-and-pruning.

Mihael Ankerst *et al.* (1999) ‘Visual classification’. Available at: https://doi.org/10.1145/312129.312298.

‌

Mihael Ankerst, Ester, M. and Kriegel, H.-P. (2000) ‘Towards an effective cooperation of the user and the computer for classification’, *Knowledge Discovery and Data Mining* [Preprint]. Available at: https://doi.org/10.1145/347090.347124.

Muhlbacher, T. *et al.* (2018) ‘TreePOD: Sensitivity-Aware Selection of Pareto-Optimal Decision Trees’, 24(1), pp. 174–183. Available at: https://doi.org/10.1109/tvcg.2017.2745158.

NUGENT, C. (no date) *California Housing Prices*, *www.kaggle.com*. Available at: https://www.kaggle.com/datasets/camnugent/california-housing-prices.

Quinlan, J.R. (1987) ‘Simplifying decision trees’, *International Journal of Man-Machine Studies*, 27(3), pp. 221–234. Available at: https://doi.org/10.1016/s0020-7373(87)80053-6.

‌

Rodriguez-Galiano, V. *et al.* (2015) ‘Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees and support vector machines’, *Ore Geology Reviews*, 71, pp. 804–818. Available at: https://doi.org/10.1016/j.oregeorev.2015.01.001.

‌

‌ ‌Schaffer, C. Overfitting Avoidance as Bias. Machine Learning 10, 153–178 (1993). <https://doi.org/10.1023/A:1022653209073>

Schneider, P. and Xhafa, F. (2022) Anomaly Detection and Complex Event Processing over IoT Data Streams: With Application to eHealth and Patient Data Monitoring. Academic Press, Cambridge. https://shop.elsevier.com/books/anomaly-detection-and-complex-event-processing-over-iot-data-streams/schneider/978-0-12-823818-9 https://doi.org/10.1016/B978-0-12-823818-9.00014-6

Scikit Learn (no date) *3.3. Metrics and scoring: quantifying the quality of predictions — scikit-learn 0.22.1 documentation*, *scikit-learn.org*. Available at: https://scikit-learn.org/stable/modules/model\_evaluation.html.

Shobha, G. and Rangaswamy, S. (2018) ‘Machine Learning’, *Handbook of Statistics*, pp. 197–228. Available at: https://doi.org/10.1016/bs.host.2018.07.004.

Song, C. and Zhang, H. (2014) ‘Discussion’, *International Statistical Review*, 82(3), pp. 359–361. Available at: https://doi.org/10.1111/insr.12060.

Song, Y. and Lu, Y. (2015) ‘Decision tree methods: applications for classification and prediction’, *Shanghai Arch Psychiatry*, 27(2), pp. 130–135. Available at: https://doi.org/10.11919/j.issn.1002-0829.215044.

‌

Soon Tee Teoh and Ma, K.-L. (2003) ‘PaintingClass’, *Knowledge Discovery and Data Mining* [Preprint]. Available at: <https://doi.org/10.1145/956750.956837>.

van den Elzen, S. and van Wijk, J.J. (2011) ‘BaobabView: Interactive construction and analysis of decision trees’, *2011 IEEE Conference on Visual Analytics Science and Technology (VAST)* [Preprint]. Available at: https://doi.org/10.1109/vast.2011.6102453.

WARE, M. *et al.* (2001) ‘Interactive machine learning: letting users build classifiers’, *International Journal of Human-Computer Studies*, 55(3), pp. 281–292. Available at: https://doi.org/10.1006/ijhc.2001.0499.

Wisconsin (Diagnostic) [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5DW2B.

‌

Wolberg, W., Mangasarian, O., Street, N., & Street, W. (1993). Breast Cancer

Wisconsin (Diagnostic) [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5DW2B.

‌

Ying, X. (2019) ‘An Overview of Overfitting and its Solutions’, *Journal of Physics: Conference Series*, 1168(2), p. 022022. Available at: <https://doi.org/10.1088/1742-6596/1168/2/022022>.

‌

‌

‌

‌

‌

‌

‌

‌

‌

‌

Appendices

Appendix A – Classification Model Output

A screenshot of a computer

Description automatically generated

Appendix B – Regression Model Output

A screenshot of a computer

Description automatically generated

Appendix C

Link to GitHub