3.1.2 Representation

# Original

The evolved trees are decision tree-like, meaning the internal (function) nodes are splitting points (binary for numeric data, and a branch for each category in categorical data), and the output of such nodes are probability vectors for each class. An example of such a tree is shown in Fig. 3. Numeric splitting points are treated as typed terminals (so values will only be crossed over between trees for a given feature, and not between features), and these values are sampled uniformly from feature ranges.

As the output of all nodes are probability vectors (as in decision trees), multiclass classification is supported directly. There are several other ways GP has been used for multi-class classification in the literature, but the output of probability vectors makes fewer assumptions (e.g. numeric outputs with class boundaries [37] assumes an ordering of classes), has better run time than others (e.g. a tree for each class [21], the run time scales with the number of classes), and also is the most interpretable due to the avoidance of complex numerical expressions or multiple trees, while also following closer to that of decision trees.

With decision trees, the input is at the root, and the output at a leaf. Traditionally with GP, the inputs are at the leaves (terminals), and the output is at the root. To get around this, data is passed in at the leaves but only functions are returned until the root node (only one branch will end up returning values for a given input), and then the functions are executed once we are at the root. For visualisation and use-cases, the two can be treated uniformly, but the details are given for clarity (i.e. in the visualisations in Section 5 class values are leaf nodes, but they are merely the majority class predicted for this branch, rather than being an evolvable node).

The trees are also designed to have the ability to construct features implicitly, from the observation that some simple patterns are not easily approximable in decision trees due to the axis-parallel splits of decision trees. Consider the simple (artificial) dataset with two features f 1, f 2, and two classes 0, 1, where the condition is if f 1 >= f 2 then class 1 else class 0. With decision trees, the resulting trees would be needlessly complex, as shown in Fig. 4. In addition to the standard binary splitting nodes discussed above, we use subtrees to construct features (as mathematical expressions), then check if these constructed features are greater than or equal to zero. This encapsulates many checks for mathematical relations, i.e.f1 - f2 >=0 is equivalent to f1 >= f2 from above. Constructed feature scan be combined using the standard mathematical operations (+,-, \*, protected /). Again, this can be important for XAI due to the dramatically simpler rules learnt, and utilising the complexity as the secondary objective prevents these constructed features from becoming overly complex.

# Condensed

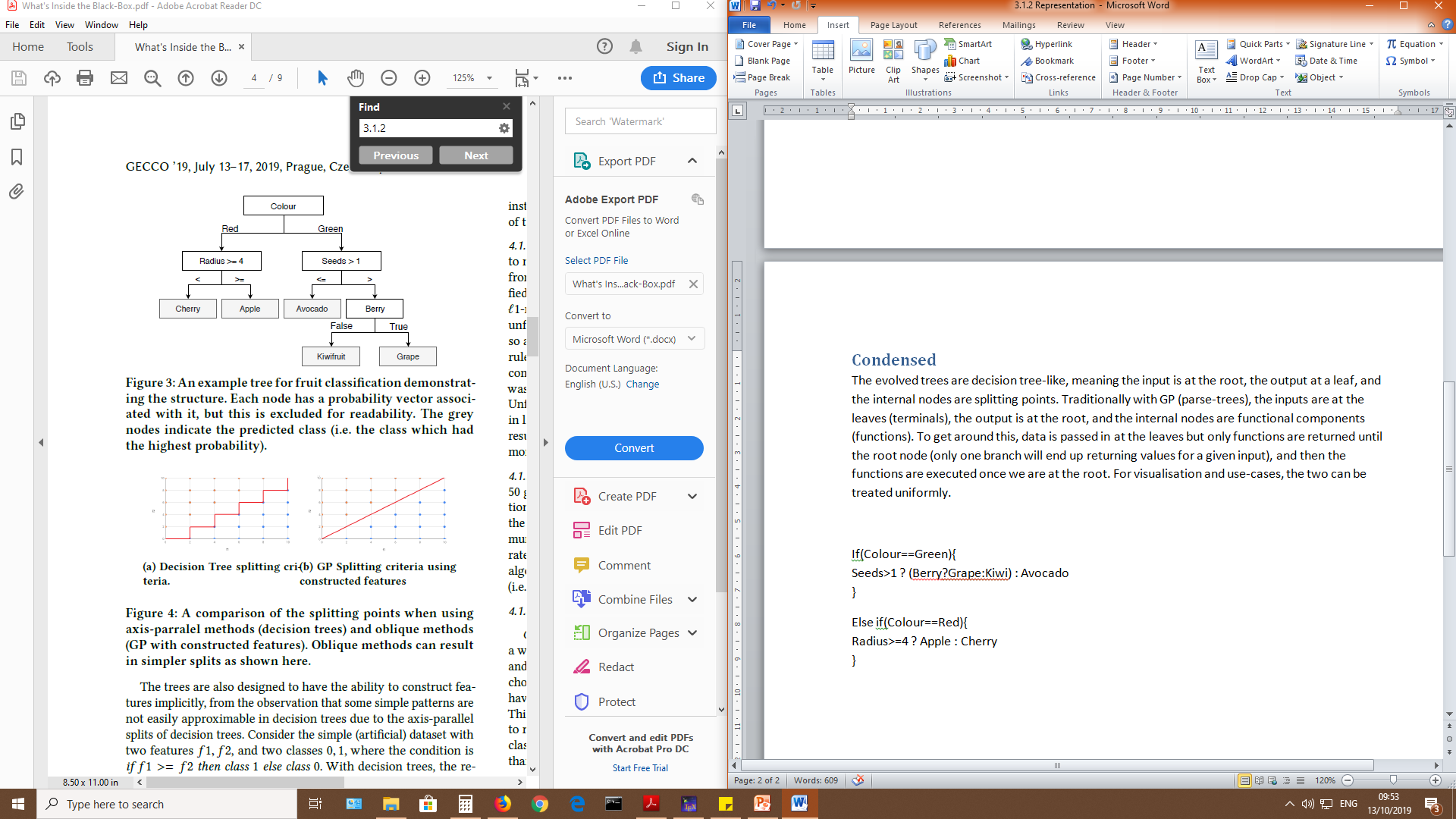
The evolved trees are decision tree-like, meaning the input is at the root, the output at a leaf, and the internal nodes are splitting points. Traditionally with GP (parse-trees), the inputs are at the leaves (terminals), the output is at the root, and the internal nodes are functional components (functions). To get around this, data is passed in at the leaves but only functions are returned until the root node (only one branch will end up returning values for a given input), and then the functions are executed once we are at the root. For visualisation and use-cases, the two can be treated uniformly.

Some simple patterns have needlessly complex representations in decision trees with axis-parallel splits. To solve this, the trees can construct features (as mathematical expressions) implicitly and check if those constructed features are greater than or equal to zero.

**For example:**

Consider the condition if f1 >= f2 then class 1 else class 0. Without constructed features the resulting tree would be complex. With constructed features we can simply use one split as f1-f2>=0.

Using constructed features is beneficial to XAI because simpler rules are learnt.



If(Colour==Green){  
Seeds>1 ? (Berry?Grape:Kiwi) : Avocado  
}

Else if(Colour==Red){  
Radius>=4 ? Apple : Cherry  
}