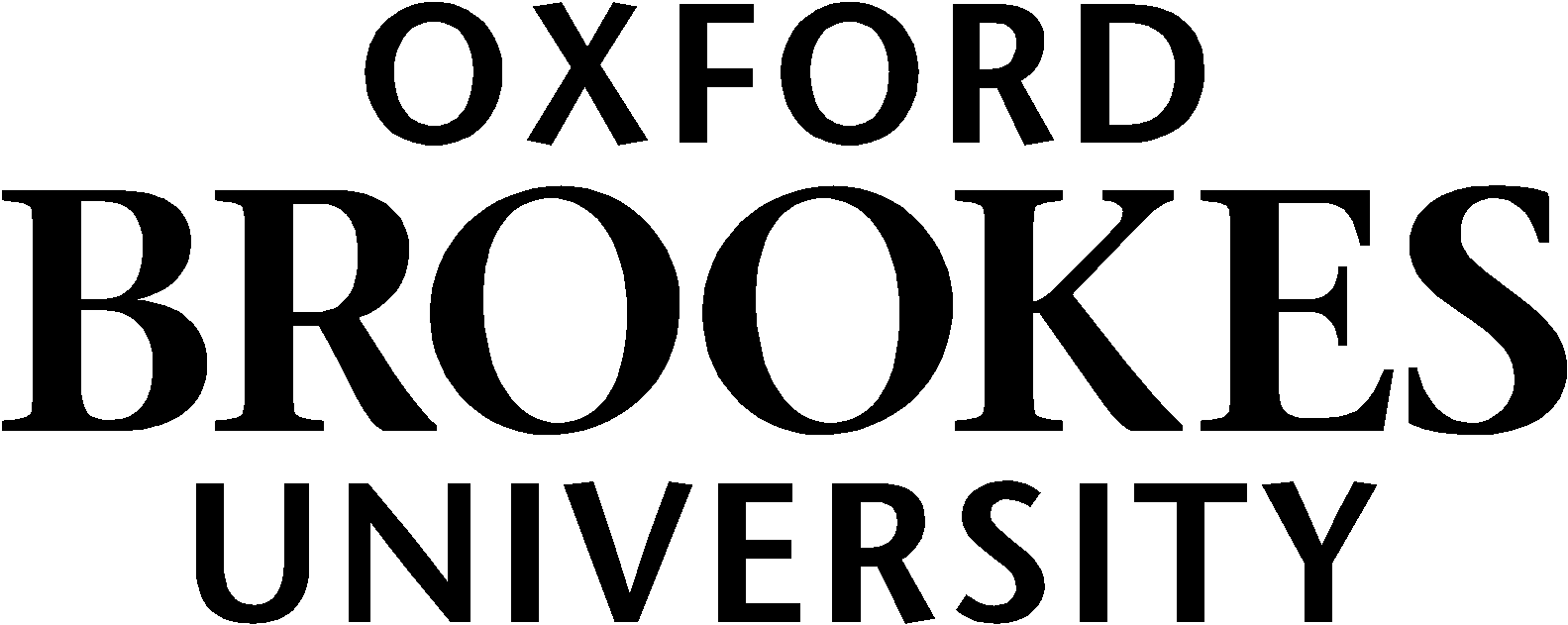
****

**Assessment cover**

|  |  |  |  |
| --- | --- | --- | --- |
| Module No: | **COMP4035** | Module title: | **Computer Science Applications** |

|  |  |  |  |
| --- | --- | --- | --- |
| Assessment number: | **Coursework** | Assessment title: | **Artificial Intelligence – Decision Tree Learning** |

|  |  |  |  |
| --- | --- | --- | --- |
| Banner assignment identifier | *CWS2WEEK6* | Due date and time**:** | **07/03/2025, 1 pm** |

|  |  |
| --- | --- |
| Estimated total time to be spent on assignment: | *30 hours per student direct work on coursework, plus 27 hours per student on independent study* |

**LEARNING OUTCOMES**

|  |
| --- |
| **On successful completion of this module, students will be able to achieve the module following learning outcomes (LOs):** |
| **LO#4** Understand principles of search, decision trees, and apply simple classification algorithms |
|  |
|  |
|  |

**Statement of Compliance *(please tick to sign)***

I declare that the work submitted is my own and that the work I submit is fully in accordance with the University regulations regarding assessments *(*[*www.brookes.ac.uk/uniregulations/current*](http://www.brookes.ac.uk/uniregulations/current)*)*

**Use of AI Tools:** You are required to use this [form](https://docs.google.com/forms/d/e/1FAIpQLSfjGiLTf7NEGMVeaZe62ufUxUs7kmw6HayzYTNKKioz_D3G2Q/viewform) to declare which AI tools you have used and how you have used them. Please complete the form and attach it to your submission as an Appendix, if you have used such tools.

Decision trees are more used than what one would think, let’s say you are to get ready in the morning and are trying to arrange an outfit with all the clothes in your wardrobe, you unconsciously create a decision tree in your head with a mismatch of your different clothes. For more practical uses though, decision trees are used to predict market changes, quality control in marketing or even in healthcare to make a diagnosis (Singh, 2023). This shows the many uses of decision trees let it be in the professional sector or in our everyday life.

Entropy in a dataset is a measure of disorder or impurity (Krishnan, 2021); in decision trees, data is split and entropy is used to find the best split in a dataset. The more splits, the more the entropy decreases making it easier to draw conclusions, however some splits will simply not add to the homogeneity leading to the entropy not changing or even increasing making it harder to draw conclusions. Given the fact that, as explained previously and as its name entails, decision trees are used to make decisions, in professional cases the best and most precise decision it is necessary for the entropy to be minimal to draw the best conclusions.

After briefly conducting research on the car dataset (see Appendix 0 for code links), we can see that there are 7 different variables, but no columns; so we must add the columns to match every variable, names of which are on the Car Evaluation Dataset website. We can also check the shape of the dataset: 1728x7.

Then check for missing variables and duplicated rows and we can see that none of either are present. Finally we check if any of the data is categorical and here even without checking we can tell the every simple piece of data is categorical. To handle the categorical data, I’ve decided to use the .map() method given fact that it is known what the different variables are for each column.

After doing so, I proceed with one last check (missing data, duplicates, info and then dataset display), to ensure that everything has been coded correctly and no data was erased or duplicated by mistake, also using df.sample() multiple times to obtain random rows from the dataset.

Now onto separating the dataset into training and testing, I’ll do 75% training and 25% testing. Now we’ll determine which class is the easiest to recognize and which is the hardest. By checking the class distribution, the feature separability and the feature correlation with the target; after emulating graphs for those three checks (see code or appendices) we can see that the unacc class is the easiest to recognize as it is the largest class in the class distribution, the most well-separated class in the feature separability graph; and the feature correlation heatmap demonstrates that the ‘Class’ feature is the easiest to recognize as well, it also has the highest correlation value.

The hardest to recognize, though hard to tell on the class distribution graph is the vgood class; it also overlaps with the good class in the separability graph making them both hard to recognize, though as said previously, the vgood is the hardest. The hardest feature to recognize is the ‘Doors’ feature as it has the lowest correlation value (absolute value).

Now after that, I’ll implement the decision tree; this has been the most challenging part of this whole project, I shall go into details on to what exactly was challenging however before that I’ll quickly go over the flow of the whole code again and the decision tree to have a full overview.

First I loaded the data and prepocessed it, I assignment column names to the car.data file and then converted all of the categorical values into numericals, then proceeded to split the dataset into training (67%) and testing (33%) sets. Then calculated the entropy and information gain to determine the best split and also what feature should be selected first to build the decision tree, after sorting the features by importance, I selected the top three best features for classification: Safety, Persons and Buying (in that order). Now onto the classification, I’ve used two different methods to go forth that I shall explain; first option (car.py): I manually defined If-Else statements to classify the top instances based on the top features, this method mimics a decision tree so though in this case I’ll call it my first decision tree it is more of a pseudo decision tree as the statements are all hard-coded thus not automatically learned.

Then I applied the classification to the dataset, compared predictions and calculated the displays accuracy, so how much it matched with the dataset. At first I used the top three features for this method but soon noticed that I could increase my decision trees accuracy by using the other features as well, so by adding depth to the decision tree, which I ultimately did until I reached a satisfactory accuracy (>67% would be considered satisfactory). With this method visualization would be pretty weak as it would be displayed in the way of a flow chart (see Appendix 4 for Flow Chart/Decision Tree representation) rather than a fully-fleshed decision tree, reason for which I decided to proceed to different ways, first method being the one I just explained and the latter being the one I shall now explain (car2.py).

The second method, I used was with the DecisionTreeClassifier import from sklearn.tree and the tree import from sklearn; I discovered this method whilst using AI to find the best possible accuracy for the decision and was suggested this method. Not only is this method easier as the classifier is in the import but it is also way more efficient as it can adapt to any change in the data, automatically finds the best splits and doesn’t require extensive hard-coded if-else statements. Then we automatically make predictions and calculate the accuracy, everything is fully automated (Breiman et al., 2017).

Finally we can visualize decision tree and review the data, this program in particular allowing it and making it easier to read and understand (Lundberg and Lee, 2017); the more depth we apply to the classifier the more precise the accuracy gets just like in the first method, doing so allows us to reach near perfect accuracy (Brieman, 2001)however it heavily reduces the clarity and legibility of the decision tree so I’ve decided to cap the max depth at 4, to optimize precision whilst also being able to clearly read the decision tree display (see appendices 5, 6 and 7 for decision tree displays at different depths); nevertheless for datasets that would not require a display adding more depths would be more efficient to yield better results (Breiman, 1996). I think it can easily be guessed why this was the most challenging and more tedious part of this whole project, with a lot of trial and error with the if-else statements ensuring that the decision tree is as accurate as possible.

In conclusion, this whole project, though challenging, was very enriching; I managed to create a full decision tree implementation and have even found a great if not best way to do it through the DecisionTreeClassifier.

For the if-else statement, I do think that a greater dataset with more features would make the tasks way more complicated, the workload gets upscaled pretty quickly with extra features especially if one is looking for maximum precision and accuracy, this kind of algorithm also mimics a decision tree but doesn’t learn from the data input. For the other method I’d say the only limitations would be one’s capacity to code and implement a dataset as the decision tree implementation is fully automated thus being way better than the if-else statement method and a great solution to its limitations (Quinlan, 1986).

References (Used AI to generate;

<https://www.mybib.com/tools/harvard-referencing-generator>):

Krishnan, S. (2021). *Decision Tree for Classification, Entropy, and Information Gain*. [online] CodeX. Available at: <https://medium.com/codex/decision-tree-for-classification-entropy-and-information-gain-cd9f99a26e0d>.

Vationventures.com. (2024). *Decision Trees: Definition, Explanation, and Use Cases | Vation Ventures*. [online] Available at: <https://www.vationventures.com/glossary/decision-trees-definition-explanation-and-use-cases>.

Singh, A. (2023). *Why: Practical Applications of Decision Trees (Part 2)*. [online] Medium. Available at: <https://medium.com/@diehardankush/why-practical-applications-of-decision-trees-ae09e04b2b16>.

Navlani, A. (2023). *Python Decision Tree Classification Tutorial: Scikit-Learn DecisionTreeClassifier*. [online] www.datacamp.com. Available at: <https://www.datacamp.com/tutorial/decision-tree-classification-python>.

*Decisiontreeclassifier* (2007) *scikit*. Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>.

Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J. (2017). *Classification And Regression Trees*. [online] Routledge. doi:https://doi.org/10.1201/9781315139470.

Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning*, [online] 1(1), pp.81–106. doi:https://doi.org/10.1007/bf00116251.

‌ Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2), pp.123–140.

Lundberg, S.M. and Lee, S.-I. (2017). *A Unified Approach to Interpreting Model Predictions*. [online] Neural Information Processing Systems. Available at: <https://papers.nips.cc/paper_files/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html>.

Breiman, L. (2001). Random Forests. *Machine Learning*, [online] 45(1), pp.5–32. doi:https://doi.org/10.1023/a:1010933404324.

Appendices:

Appendix 0: Code URLs

Code for car.py (if-else statements): <https://pastebin.pl/view/f73764c2>

Code for car2.py (DecisionTreeClassifier): <https://pastebin.pl/view/2b29b2b2>

Car Dataset: <https://pastebin.pl/view/f3be091a>

Car Names: <https://pastebin.pl/view/8984043c>

Appendix 1: Class Distribution Graph:

A graph with blue squares

AI-generated content may be incorrect.

Appendix 2: Feature Separability Graph:

A graph of a graph

AI-generated content may be incorrect.

Appendix 3: Feature Correlation Heatmap:

A screenshot of a chart

AI-generated content may be incorrect.

Appendix 4: Flow Chart

A diagram of a computer program

AI-generated content may be incorrect.

Appendix 5: Decision Tree (max depth=4)

A diagram of mathematical equations

AI-generated content may be incorrect.

Appendix 6: Decision Tree (max depth=7)

A diagram of a network

AI-generated content may be incorrect.

Appendix 7: Decision Tree (max depth=10)

A diagram of a network

AI-generated content may be incorrect.