Problem Set 1

Jacob Mongold

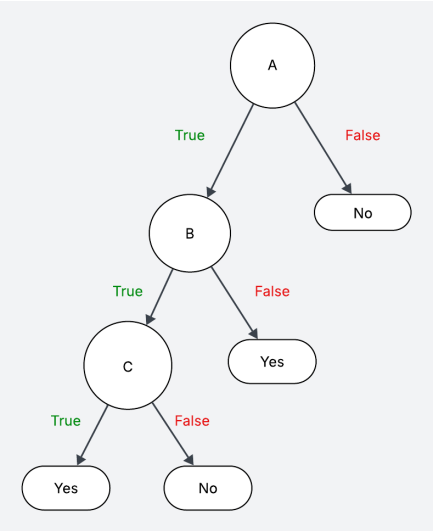
September 2025

Problem 1: Decision Trees for Boolean Functions

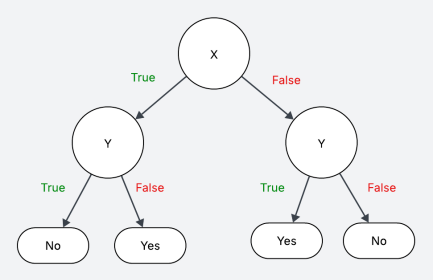
Q1: Boolean Expressions

Below are the decision trees for the given Boolean expressions.

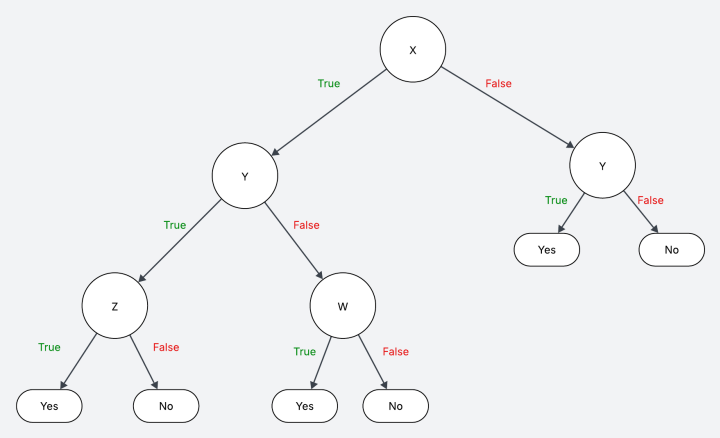
1. $A \wedge \bar{B} \wedge C$



2. $X \wedge \bar{Y} \vee \bar{X} \wedge Y$



3. $X \wedge Y \wedge Z \vee X \wedge \bar{Y} \wedge W \vee \bar{X} \wedge Y$



Q2: Root Node Selection using Information Gain and Gini

Formulas: Entropy and Gini Index

$$H(D_i) \;=\; -\sum_{c_i \in \{C\}} P(c_i)\log_2 P(c_i), \qquad \text{Gini}(D_i) \;=\; 1 - \sum_{c \in \{C\}} p_c^2.$$

For feature f with levels ℓ:

$$\text{IG}(D_i, f) \;=\; H(D_i) - \sum_{l \in \{levels(f)\}} \frac{|D_{i,l}|}{|D_i|}\, H(D_{i,l})$$

Parent Node

Dataset size: $|D| = 14$
Positives: 8, Negatives: 6

$$p_+ = \tfrac{8}{14}, \quad p_- = \tfrac{6}{14}$$
$$H(D) = 0.9852, \qquad \text{Gini}(D) = 0.4898$$

Feature: League

Counts (T,F): SerieA $(3, 2)$, LaLiga $(4, 1)$, Premier League $(1, 3)$

$$H(\text{SerieA}) = 0.9710, \quad \text{Gini(SerieA)} = 0.4800$$
$$H(\text{LaLiga}) = 0.7219, \quad \text{Gini(LaLiga)} = 0.3200$$
$$H(\text{PL}) = 0.8113, \quad \text{Gini(PL)} = 0.3750$$

Weighted values:

$$H_{\text{League}} = 0.8364, \quad \text{IG} = 0.1488$$
$$\text{Gini}_{\text{League}} = 0.3929, \quad \text{Gini} = 0.0969$$

Feature: Position

Counts (T,F): CF $(5, 1)$, LW $(1, 3)$, RW $(2, 2)$

$$H(\text{CF}) = 0.6500, \quad \text{Gini(CF)} = 0.2778$$
$$H(\text{LW}) = 0.8113, \quad \text{Gini(LW)} = 0.3750$$
$$H(\text{RW}) = 1.0000, \quad \text{Gini(RW)} = 0.5000$$

Weighted values:

$$H_{\text{Position}} = 0.7961, \quad \text{IG} = 0.1891$$
$$\text{Gini}_{\text{Position}} = 0.3690, \quad \text{Gini} = 0.1207$$

Feature: Preferred Foot

Counts (T,F): Left $(2, 5)$, Right $(6, 1)$

$$H(\text{Left}) = 0.8631, \quad H(\text{Right}) = 0.5917$$
$$\text{Gini(Left)} = 0.4082, \quad \text{Gini(Right)} = 0.2449$$

Weighted values:

$$H_{\text{Foot}} = 0.7274, \quad \text{IG} = 0.2578$$
$$\text{Gini}_{\text{Foot}} = 0.3265, \quad \text{Gini} = 0.1633$$

Feature: Capped

Counts (T,F): yes $(4, 3)$, no $(4, 3)$

$$H(\text{yes}) = H(\text{no}) = 0.9852, \quad \text{Gini(yes)} = \text{Gini(no)} = 0.4898$$

Weighted values:

$$H_{\text{Capped}} = 0.9852, \quad \text{IG} = 0$$
$$\text{Gini}_{\text{Capped}} = 0.4898, \quad \text{Gini} = 0$$

Summary Table

Feature	**IG**	**Gini**
League	0.1488	0.0969
Position	0.1891	0.1207
Preferred Foot	0.2578	0.1633
Capped	0	0

Root Choice: The best root node is Preferred Foot, chosen by both Information Gain and Gini reduction.

Problem 2: Implementing the Decision Tree Algorithm

Q3: Describe your implementation in one paragraph.

My implementation builds a decision tree from scratch using NumPy and some Pandas for handling the data. I defined functions to calculate entropy and Gini impurity, along with an information gain function to evaluate splits. The build-tree function handles the recursive structure of the tree, where each node either becomes a leaf with a majority class prediction or splits again on the best feature and threshold. I created simple classes for Leaf and DecisionNode to keep track of predictions and splits, and then wrapped everything in a DecisionTree class so I could fit the model and run predictions more easily. I tested the tree on training, development, and test sets, trying different depths and criteria to see how the accuracy changed.

Q4: Discuss any challenges you faced while implementing your decision tree.

This problem set has been the most challenging assignment I have worked on because of the amount of material I needed to learn and relearn along the way. At my level of coding, I am still getting used to how arrays are shaped, indexed, and concatenated, so one of the main problems was structuring the data correctly. If the array was not in exactly the right shape or concatenated correctly, the splitting function would not work and the code would either throw errors or just fail silently (which occurred more often than actual error messages). Some of the real difficulty came from implementing recursion in the build tree function. At first, I did not set a maximum depth or minimum samples, which meant the model was bound to overfit because it was memorizing the data. Then figuring out the base cases and making sure the function would actually return a leaf instead of just running forever took me a while. Some of the other issues are documented in comments within the code, while others are not. I tried to implement the chi-squared method, but that just plagued me with bugs and I was running out of time. Overall, the assignment was especially difficult because I do not have a lot of prior experience coding in Python or using its libraries. Getting the code fully working required more than 12 hours of effort.

Q5: Explain how you overcame the challenges you described in Q4.

I overcame these challenges mostly through trial and error, checking documentation, and slowly piecing things together. For the array problems, I voiced my problems to one of my roommates. He suggested that I start printing out shapes and slices of the data to see what was actually going wrong and learn how things work (he was just going to give me the answer but I wouldn't learn anything if he did that). Then, once I saw where they were going wrong, I restructured how I was building and concatenating the arrays until the splits worked. To deal with overfitting, I added maximum depth and minimum samples as stopping conditions. This finally got the recursion under control. Figuring out the base case for the build tree function took me a lot of back and forth, but after writing the same 27 lines of code several times over, it finally started returning leaves where it should. Even though I could not get the chi-squared method fully working, I learned from some of the bugs I hit and focused on getting the main tree correct. I think that if I built the chi-squared while I was writing the entire thing, I could have been more successful with it. I tried adding it at the end and I just kept breaking everything. In the end, I worked through each issue one by one and built enough of an understanding of Python and Numpy to get the code running.

Problem 3: Use your decision tree to develop a model for WDBC

Q6: Summarize the dataset paper you read and discuss two key observations.

The WDBC dataset contains features computed from digitized images of fine-needle aspirates of breast masses. Each individual image is processed to extract 30 features that describe the characteristics of the cell nuclei. A few example features are radius, texture, perimeter, area, and smoothness. The dataset has 2 classes: malignant (M) and benign (B).

Key observations

1. The researchers point out that many of the 30 features are correlated because they are all derived from measurements of similar cell properties. This means that not all features contribute unique information. This can affect the performance of algorithms that are sensitive to redundant inputs.

2. The dataset was designed to support research into reliable computer-aided diagnosis. The goal was to reduce faults in human judgment while evaluating fine-needle aspirates. This is why classification accuracy is so important. If the model classifies a malignant tumor as benign, the patient outcome could potentially be disastrous.

Q7: Performance results on the test set

Model	Accuracy	Error	Precision (M)	Recall (M)
Entropy (depth=3)	0.947	0.053	0.974	0.881
Gini (depth=5)	0.965	0.035	1.000	0.905
Majority baseline	0.632	0.368	0.000	0.000

Table 1: Test set performance of best decision trees and a majority-class baseline. Malignant (M) is treated as the positive class.

The results showed that entropy at depth 3 achieved 94.7 percent accuracy with high precision of 0.974 but recall of 0.881 was a bit lower, meaning that it still missed some malignant cases. Gini at depth 5 performed best overall with 96.5 percent accuracy, perfect precision of 1.000 and a better recall of 0.905. The majority baseline was much worse with only an accuracy of 63.2 percent, and 0 for precision and recall. This shows that using a decision tree to learn patterns in the features is meaningful, while the trivial baseline completely fails to identify malignant cases.

Q8: Discuss if precision or recall is most critical (most important) as a performance metric for this problem

Precision in this case measures how often the model's malignant predictions are actually correct and whether any benign cases are mislabeled as malignant. Recall shows how many malignant cases the model fails to identify. For a medical problem like cancer, recall is the most important metric. It is much better to maximize recall so that as many malignant cases as possible are caught, even if that lowers precision and leads to some benign cases being flagged. A false alarm can always be reviewed by a medical professional, but missing a malignant tumor could mean letting cancer grow untreated.

Q9: Explain your tuning procedure with the dev (tuning) set and what you learned from this part of the problem.

For tuning, I used the dev set to compare different tree depths and splitting criteria. I tested entropy and gini at depths 3, 5, and 10. I then looked at which settings gave the best balance between dev and test performance. What I found was that

shallow and medium depth trees worked better because they generalized more than the deeper trees, which started to overfit. For example, entropy at depth 3 gave the highest dev accuracy, but gini at depth 5 ended up performing the best overall since it had the strongest test accuracy and recall. From tuning, I learned that setting limits on depth is very important. Also that both entropy and gini are effective, but in this trial gini was a little more stable across different splits.

Problem 4: Compare with an implementation in an ML library

Q10: Decision-tree Visualization

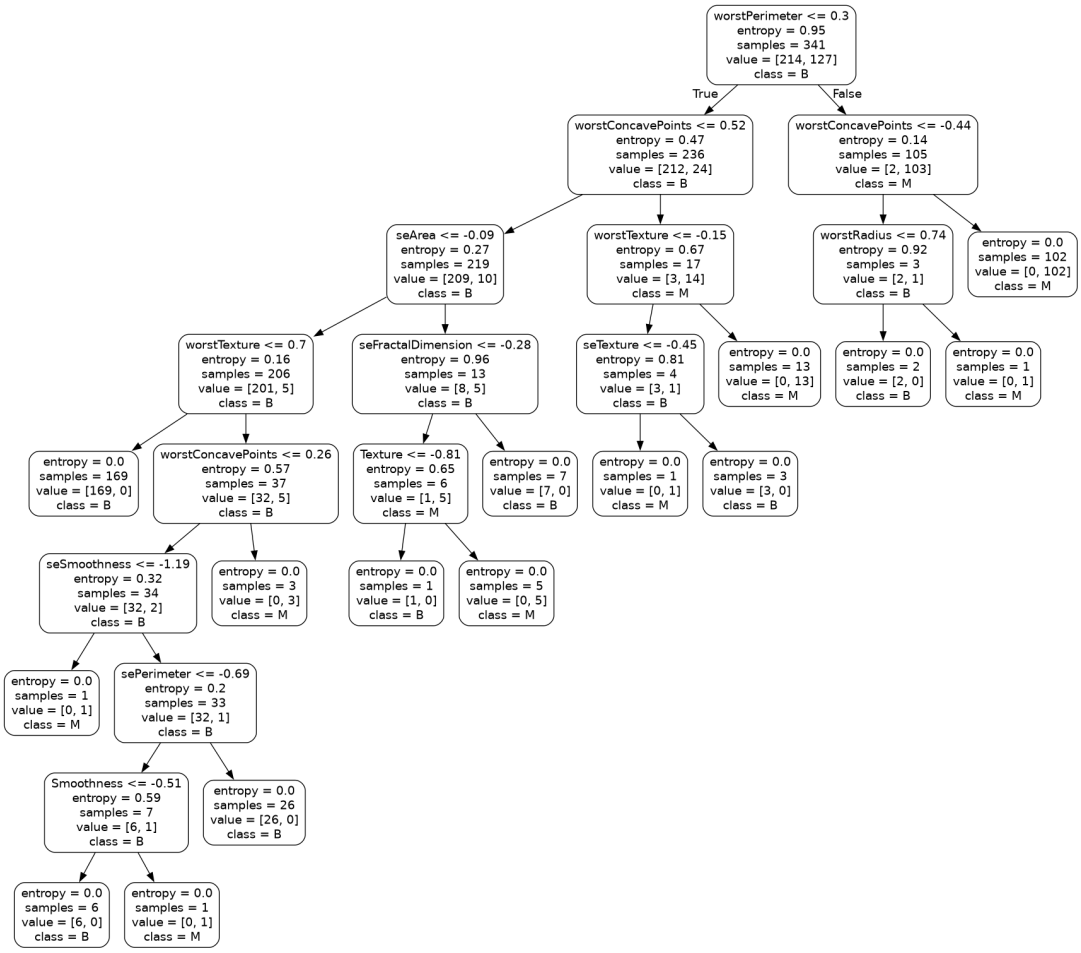


Figure 1: Held out test-set and output performance metrics decision tree

Q11: Discuss how your best performing informativeness metric compared with the best-performing result in your experiment.

The best-performing metrics for each model are different. We are going to treat malignant as the positive class in this case as well. Starting with my model, Gini performed the best at depth 5 with 96.5 percent accuracy, perfect precision of 100 percent, and a recall of 90.5 percent. The scikit-learn tree that performed the best came from testing depths between 3 and 50. That model selected entropy with a max depth of 6 as the best parameters. Its global accuracy was 96 percent, with a precision of 97 percent and a recall of 93 percent. Even though my model had slightly higher precision, the scikit-learn model had stronger recall and a better overall balance between the two metrics. In the context of cancer detection, recall is more important than precision, so the scikit-learn model is ultimately the better choice because it misses fewer malignant cases while still keeping precision high.

Q12: Discuss how your best performing informativeness metric compared with the best-performing result in your experiment.

My implementation and the scikit-learn implementation of decision trees are similar in that they both rely on impurity measures like Gini or entropy to recursively split nodes and build the tree. Both models follow the same overall process of finding the best split at each node and applying stopping conditions. After that, the similarities end. My tree was coded from scratch and only supported a few hyperparameters like depth, criterion, and minimum samples. Writing it also required me to manually handle recursion, stopping conditions, and data handling, which made it more prone to bugs. The scikit-learn model comes with many more hyperparameters. It includes options for pruning, controlling the number of features to consider when splitting, and even a random state function to reproduce or vary results. It also handles most of the complexity automatically through its built-in functions. Another big advantage is that scikit-learn provides tools for evaluation and visualization out of the box, which made testing and interpreting the model much easier. My implementation worked and gave good performance, but the scikit-learn version is more robust, efficient, and user-friendly. It does more with less code.

Q13: Performance of binned vs. non-binned data

Table 2: Comparison of Binned vs. Raw Normalized Data Performance

Data Type	**Accuracy**	**Error**	**Precision (M)**	**Recall (M)**
Binned	0.965	0.035	1.000	0.905
Raw normalized	0.960	0.040	0.970	0.930

The binned data achieved slightly higher accuracy (0.965 vs. 0.960) and perfect precision (1.000), but this came at the cost of lower recall (0.905 vs. 0.930). The raw normalized data performed more evenly across both precision and recall, making a more balanced classifier.

Q14: Explanation

Turning continuous values into bins throws away information, which can hurt accuracy. At the same time, binning can make a model simpler and less likely to overfit, because it limits the number of possible split points. The binned data led to a simpler tree that favored precision, while the raw data gave scikit-learn the ability to split more precisely and balance precision and recall better.

Q15: Additional Hyperparameter Observations

Table 3: Performance of Additional Hyperparameters

Configuration	Accuracy	Precision (M)	Recall (M)	F1-score (M)
Random state = 58	0.94	1.00	0.83	0.91
Random state = 43	0.96	0.97	0.93	0.95
Min split/Min leaf	0.93	0.90	0.90	0.90

I tested min samples split and min samples leaf along with different random states. The random state matters because it controls how the algorithm breaks ties and how the data is shuffled before building the tree. Different seeds can give you slightly different splits, which can change the balance between precision and recall. With random state 58 the model had perfect precision for malignant cases (1.00) but recall dropped to 0.83, so it was missing more true positives. With random state 43 the performance balanced out with accuracy hitting 0.96, precision was 0.97, and recall was 0.93. When I used grid search over the split and leaf parameters, the tree got simpler and the accuracy dipped to 0.93 with both precision and recall for malignant cases at 0.90. This shows that bigger split and leaf values make the tree shallower and reduce overfitting, but it comes at the cost of sensitivity. Smaller values let the tree capture more malignant cases but risk overfitting. For this dataset, the best balance came from random state 43 where the tree was complex enough to catch patterns without being too deep.