IDAI 610
Problem set 1: Decision Trees for Clinical Data

Objective: In this problem set, you will implement the standard decision tree algorithm including two ways to implement node splitting, for a comparison of their results. You will also use your decision tree implementation to create a predictive model with the Wisconsin Diagnostic Breast Cancer Data, using training, tuning, and test sets. Subsequently, you will analyze and discuss the results. In a final task, you will compare with a decision tree implementation in a standard machine learning package named `scikit-learn` and explore additional hyperparameters of decision trees as well as decision-tree visualization.
Submission Instruction: Submit your report and notebook(s) as `ps1-[LastName]` as in this example: `ps1-alm.[zip|tar.gz]` , in the assigned assignment dropbox in our myCourses website. Remember to include your written report, code, and a succinct readme explaining how to run your code. Clearly indicate which question you are responding using the format **Q*n***.

Wisconsin Diagnostic Breast Cancer (WDBC) Data

The Wisconsin Diagnostic Breast Cancer Data[1] is a benchmark dataset from the domain of machine learning in health. It was collected by researchers from the University of Wisconsin Hospitals, Madison. The dataset contains features extracted from digitized images of fine needle aspirates (FNA) of breast masses. These features are used to predict whether a breast mass is **benign** (non-cancerous) or **malignant** (cancerous). **Note**: You will use reformatted data provided with this assignment, instead of the original data release.

Please start this assignment by reading the paper: W. Nick Street, W. H. Wolberg, and O. L. Mangasarian "Nuclear feature extraction for breast tumor diagnosis", Proceedings of SPIE 1905, Biomedical Image Processing and Biomedical Visualization, (29 July 1993); `https://doi.org/10.1117/12.148698`.

Problem 1: Decision trees for Boolean functions

Q1: Draw a Decision Tree based on Boolean functions. (2 points)

Example decision tree for $X \wedge Y$, where X and Y are the binary features and **Yes** and **No** are the labels:

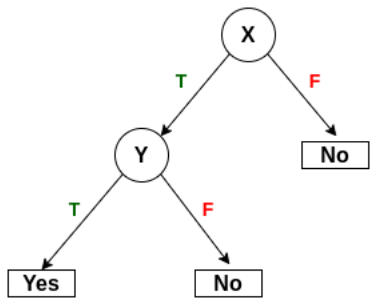

Figure 1: Decision tree for $X \wedge Y$

Based on the example draw a decision tree each for following Boolean expressions:

[1]Original dataset at: `https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic`.

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 (4 points)

Player	League	Position	Preferred foot	Capped	Shortlisted
A	SerieA	CF	Left	yes	True
B	LaLiga	LW	Right	no	True
C	LaLiga	CF	Right	yes	True
D	PremierLeague	CF	Left	yes	True
E	PremierLeague	LW	Left	yes	False
F	SerieA	RW	Left	yes	False
G	SerieA	CF	Right	no	True
H	PremierLeague	LW	Left	no	False
I	SerieA	RW	Right	no	True
J	LaLiga	RW	Left	yes	False
K	LaLiga	RW	Right	no	True
L	PremierLeague	CF	Right	no	False
M	LaLiga	CF	Right	yes	True
N	SerieA	LW	Left	no	False

Table 1: Data the new soccer club's sports director has for shortlisting forward players A through N.

Based on Table 1, compute the selection procedure of the root decision node (considering the 4 features) using standard Information Gain and Gini. Show all the intermediary calculation steps.

Problem 2: Implement the decision tree algorithm (18 points)

In this problem, you will implement a recursion-driven, top-down, decision tree from scratch, following the pseudocode of the algorithm described in class or in section 19.3 in R&N for solving a Boolean classification problem with binary labels. **You may not use an existing library implementation of decision trees.** (Using a helper library like `numpy` is allowed and requires you to include a `requirement.txt` in your submission; see the corresponding file already included for problem 3.) Your code should be well-documented. We recommend a Jupyter notebook with comments included directly, but coding with IDLE or another environment and *.py* files is also allowed. You should test that your code runs on the `cs.rit.edu` server enviornnment before you submit.

Your decision tree implementation should include two attribute/node splitting techniques, based on informativeness, and allow the user to select which one to use:

1. **Entropy and Information Gain**
   Entropy is a measure of heterogeneity (uncertainty) in data. Higher entropy implies less information, and vice-versa. Entropy quantifies the uncertainty about the class distribution within a node in decision tree. A node with low entropy comprises mostly one class while a node with high entropy contains a broader distribution across classes. The entropy of a node is given by:

$$Entropy(D_i) = -\sum_{c_i \in C} P(c_i) \times \log_2 P(c_i) \tag{1}$$

   where D_i is the set of instances within the node under consideration, c is the number of classes, and $P(c_i)$ is the probability of instances belonging to class i.

   Information Gain (IG) is a way to quantify the quality of split based on entropy. It is used to evaluate the effectiveness of a feature in reducing entropy. You will implement IG as the difference between the entropy of the parent node and the weighted average entropy of the child nodes after the split. The feature with the highest IG is selected as the splitting feature. The Information

Gain $IG(D_i, f)$ of selecting an feature f with levels l, relative to set of instances D_i in the parent node is given by:

$$IG(D_i, f) = Entropy(D_i) - \sum_{l \in levels(f)} \frac{|D_{i,l}|}{|D_i|} Entropy(D_{i,l}) \tag{2}$$

where $D_{i,l}$ represents the subset of instances that have value l for feature f.

2. **Gini**
   The Gini index measures the likelihood of a randomly selected instance being classified incorrectly. (It can be particularly valuable for continuous features, e.g., in CART or Classification and Regression Trees.) It too quantifies the impurity or uncertainty of values in a dataset. And similarly, a lower Gini index indicates a homogeneous distribution while higher indicates heterogeneous mix of instances within a node. The Gini index is given by:

$$Gini = \sum_{c_i \in C} P(c_i)(1 - P(c_i)^2) \tag{3}$$

$$= 1 - \sum_{c_i \in C} P(c_i)^2 \tag{4}$$

Finally, you must implement tuning and testing procedures (functions) as part of your decision tree implementation. You will use these in problem 2 so we recommend you read that part before beginning your own implementation.

Your report on this problem should discuss **Q3:** your implementation in one paragraph, **Q4:** any challenges you faced, and **Q5:** how you overcame them.

Extra credit (8 points): Implement the pruning element of the decision tree with χ^2 pruning, described in section 19.3.4 in R&N for 8 extra bonus points.

Problem 3: Use your decision tree to develop a model for WDBC (14 points)

For this problem, you will use the *train*, *dev* (standing for dev-test or validation or tuning set), and *test* sets, available in CSV files provided in myCourses. You will seek to train, tune (using the data partition called dev), and test your own decision tree that classifies tumor nuclei into two classes: **malignant** (M) and **benign (B)**. Thus, it is a Boolean classification problem.

The tuning phase will focus on selecting the best-performing node-splitting criterion (Gini or IG). If you implemented χ^2 pruning for problem 1, you could gain **2 extra points** by considering it, i.e., setting χ^2 pruning to active or not active (on or off), in your tuning process.

Finally, you will test your decision tree's predictions on the held-out test data. The test set aims to approximate data seen in deployment. It may not be used before predicting your final results. *Thus, you may NOT review the test set during development, or re-tune to the test set.*

A description of columns in WDBC.

1. **Binary class labels:**
   The column *Diagnosis* includes diagnostic target label where M = malignant and B = benign.

2. **Features used for class prediction:**
   There are 30 distinct features, which are based on 10 continuous (real-valued) features extracted from cell nucleus information. As noted in the website of the original dataset paper, these ten measures were as follows - quoting here from the dataset website:

   1) **Radius**: mean of distances from center to points on the perimeter
   2) **Texture**: standard deviation of gray-scale values
   3) **Perimeter**
   4) **Area**
   5) **Smoothness**: local variation in radius lengths
   6) **Compactness**: $\frac{perimeter^2}{area} - 1.0$

7) **Concavity**: severity of concave portions of the contour
8) **Concave points**: number of concave portions of the contour
9) **Symmetry**
10) **Fractal dimension**: coastline approximation - 1

Aggregate values including the mean, standard error, and largest (mean of the three largest values) termed *worst* of each of the ten features were computed per medical image, resulting in 30 (3×10) features. For instance, in the data files, the feature *Radius* refers to the mean radius, whereas feature *seRadius* refers to the radius standard error, and *worstRadius* refers to the worst/largest radius, as just explained.

Discretization:
All the actual feature values are continuous. However, a standard ID3 decision tree implementation expects categorical features. Thus, there is a need to *discretize* or *bin* the continuous, real-valued features in the original dataset. This has been done for you, with values binned into six categorical level (***l1, l2, l3, l4, l5, l6***). For discretization, each feature column was first Z-score normalized:

$$Z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j} \tag{5}$$

where x_{ij} is the j^{th} feature (column) of the i^{th} image instance (row), μ_j and σ_j are the mean and standard deviation of j^{th} feature respectively, and Z_{ij} is the Z-score normalized value of x_{ij}. The normalized values were then referred to as six levels (l) of the feature f, coded as:

$$\text{Code}(Z_{ij}) = \begin{cases} l_1 & \text{if } Z_{ij} < -2\sigma_j \\ l_2 & \text{if } -2\sigma_j \leq Z_{ij} < -\sigma_j \\ l_3 & \text{if } -\sigma_j \leq Z_{ij} < \mu_j \\ l_4 & \text{if } \mu_j \leq Z_{ij} < \sigma_j \\ l_5 & \text{if } \sigma_j \leq Z_{ij} < 2\sigma_j \\ l_6 & \text{if } Z_{ij} \geq 2\sigma_j \end{cases}$$

You can use the discretized data in `./Final_data/wdbc_{train/dev/test}.csv`. (Details on the discretization process can be found in *Discretization_prepartion.ipynb* and reviewing it is optional.)

Extra credit (5 pts.)
You can discretize the original continuous data in `./Final_data/wdbc_{train/dev/test}_raw.csv` into categorical data yourself using (a) your own approach, or (b) the procedure briefly explained in R&N in section 19.3.5 (p. 664). Please describe the process you used in plain English and do your best to express it formally as well, as examplified above.

In your report, answer these questions: **Q6:** summarize the dataset paper you read and discuss two key observations. Then, **Q7:** provide the performance results on the test set in a five-column table with accuracy, error, precision, and recall, using rows for Gini and IG, respectively. Your table should also include a comparative baseline result for a trivial majority class classifier that only uses the most frequent class in the training data to label all instances. Note that for precision and recall, you will need to logically decide what you treat as the positive class (to determine true and false positive predictions, respectively). Additionally, **Q8:** discuss if precision or recall is most critical (most important) as a performance metric for this problem. Finally, **Q9:** explain your tuning procedure with the dev (tuning) set and what you learned from this part of the problem.

Problem 4: Compare with an implementation in an ML library (12 points)

For this problem, you will use the Jupyter notebook provided in myCourses and compare it with your previous results, study additional decision tree hyperparameters, and visualize the resulting decision tree.

Explore the notebook and the differences you observed in the performance between `scikit-learn` implementation and your implementation.

In your report, **Q10:** include the decision-tree visualization and **Q11:** discuss how your best-performing informativeness metric compared with the best-performing result in your experiment. Overall,

Q12: elaborate on the similarities and differences between your implementation and that of `scikit-learn`. Additionally, **Q13:** compare and contrast the performance of the binned versus non-binned (normalized) data and report if there is a difference in performance between the two, and **Q14:** speculate why that may be?

Finally, review the documentation on decision trees for the library at `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html` and then explore two additional decision tree hyperparameters that you did not yet consider such as a maximum depth stopping criterion, the minimum number of instances for a split, or the minimum number of instances at a leaf node. Discuss **Q15:** your observations, e.g., based on visualized evidence or performance results.