Assignment 2.1 Exercises

For this assignment, you will conduct programming tasks in Google Colab and answer provided questions in a Google Doc or MS Word document. You will submit **2 documents** to Blackboard:

1. A PDF file converted from Google Colab. Questions that require you to complete answers in **Google Colab** are highlighted in blue.
2. A PDF file converted from your Assignment 2.1 Template Word Doc (document is linked in Blackboard assignment prompt). Questions that require you to complete answers in your **Assignment 2.1 Short Answer Template** are highlighted in green.
    a. This template has a copy of all short answer questions for this assignment.

Questions:

1. **Ranking Problem:** Consider the following ranking problem. You have data from pairs of players who are playing two-player games against each other (for example, chess). For each player, you have calculated a score, r_a for player a and r_b for player b. The probability that player a will beat player b in a game is given by $\frac{r_a}{r_a + r_b}$. The scores are positive numbers, and the greater the score assigned to a player, the better the player is: they are more likely to win.
    - This is an example of a ranking problem: the scores tell us which users are more likely to beat each other. If we had to predict the results of a tournament between players, we could use the probabilities to estimate the most likely outcome of the tournament.
    - The simple model described here is known as the Bradley-Terry model and is similar to the ELO rating system used in chess or TrueSkill, used by Microsoft to rank Xbox live users. TrueSkill has very detailed documentation if you are interested in learning more about how the algorithm works, available from the TrueSkill website. This information is not necessary to answer the questions in this assignment.
    - When we train the ranking model, we are given two sets of inputs: for every game, we know which two players played in it, and for every game, we have the outcome of that game (win, lose, or tied).
    - When we train the ranking model, we adjust the values of the scores until the probabilities output by the model is a good fit for the data. After training, the scores give accurate predictions of the games used to train the ranking model.

Answer the following questions in the **Assignment 2.1 Short Answer Template Word Doc**.

If our goal is to predict the outcome of the next match between a pair of players:

- Is predicting the outcome of a match between two players, given their scores, a regression or classification problem? Explain your answer in one sentence.
- Are the scores, r_a, r_b, and so on examples of hyperparameters, parameters, or labels? Explain your answer in one sentence.

2. **Decision Trees:**

In these sections (2.1, 2.2, and 2.3 below), you will develop decision tree classifiers for three datasets (shelter_data.csv, text_data.csv, and audio_data.csv). Do not change any of the random seeds or the training/test split for these data. You can save a lot of work by re-using your code as you progress through the different sections. Make sure to have sensible variable names, comments, and clean code. As you progress through this and future assignments, you may reuse some common blocks of code. It's worth it to put the effort in now to build reusable, solid code, including readable variable names and comments. In 2.1, 2.2, and 2.3 below, you will go through the following steps:

1. Build the first classifier using default settings.
2. Report accuracy.
3. Experiment with hyperparameter to see how it affects accuracy.

2.1: Animal Control/Shelter Data: Complete this in **Google Colab.**

Building on the example in the lecture, develop a classification model for the shelter dataset (located in your assignment prompt in Blackboard).

- First, look at the starting code to see how to train a decision tree using scikit-learn. Make sure you don't change the random seed or any other parameters.
- Load the shelter_data.csv file - follow the sample code to see how to format and load from this file.
- Train a new decision tree using
  - short_tree = tree.DecisionTreeClassifier(max_depth=3)
  - And then, use the scikit-learn functions to train and test this decision tree, including plotting the confusion matrix (as above) and plotting the smaller decision tree.
- Vary the decision tree depth in a loop, increasing depth from 3 to 10. Make a plot of training and test set error as the decision tree depth is varied.
- On the x-axis should be your depth parameter and on the y-axis the error. Plot both the training and test set error as two separate curves. Label your plot and

give it a legend. Write a small description (two sentences) in a markdown cell below this plot, explaining what it is.

- Use pruning to control the complexity of the decision tree. Pruning uses a complexity parameter to control tree size and remove unnecessary nodes. Some documentation for how scikit-learn implements pruning is here.
- The writing on that page is not super-clear, but you should be able to extract the following information from that page:
    - Ccp_alphas is a parameter that controls complexity
    - You should see how to get a range of values for this parameter and how to use it to control the complexity of a decision tree, as we did for depth in the previous example.
    - The documentation shows how accuracy can change as a function of this ccp-alpha parameter (you can extract and run this code on your own data).
- Plot a pruned decision tree. A good value for ccp_alpha is the mean, about 0.0007
- Compare the accuracy of both the pruned and unpruned trees on the test set. Display a confusion matrix for both the original and pruned decision tree.
- Formatting what you hand in is important, and one-way practicing data scientists communicate is by sharing the notes - take some time to familiarize yourself with markdown.
- To give some structure to your assignments: short sections of text and section headings. For this assignment, use the existing template as a guide: keep in mind that you are writing this document to communicate, and make it easier for your audience (and grader!) to find information in the file.
    - For example, In the previous question, when you use pruning to create a new tree, make sure to call attention to and label both of the confusion matrices: it could be as simple as using a heading in a markdown cell to indicate the pruned and unpruned trees.
- Use matplotlib - display inline in your code, and export it to a PDF. Make sure the resolution and settings are such that the text of all plots is easily readable!

2.2: Text Data: Complete this in **Google Colab.**

Download the file text_data.csv file (located in your assignment prompt in Blackboard). Each feature in this dataset is how often a word appears in the original text.

- In this section, you will again train and test a decision tree, this time using text data. The starting template Jupyter notebook will get you started with this data.
- Then you will write code (you can re-use a lot from the previous question) to vary the depth of the tree you train on the text dataset. To get the plot of accuracy, you

can re-use your code from 2.1. Make sure to set the test size to 0.10. Plot the accuracy (train and test) as you vary the depth of the tree from 1 to 15.

- You can fine-tune the data by choosing which words to include. The provided code chooses rare words, and unfortunately, as coded, it chooses hapax legomena - words that only appear once in the corpus. Try this, and you should see you are getting poor classification accuracy. What does the plot you get by varying depth look like in this case? Write code to vary the depth of a decision tree trained on the text dataset, as before, but this time only using the rare words in the dataset.
- Hint: You can re-use the same block of code as before. You might be overwriting variables, which would give you confusing errors which are difficult to debug.
- Fix this code to choose the most frequently used words instead. Study the code provided in the previous section and modify the code to choose words that appear more than 100 times in the entire dataset. Calculate the test set accuracy when using only the words appearing more than 100 times, compared the accuracy when using all the words.
- Again re-use the same block of code to vary decision tree depth.

Answer these questions in your **Assignment 2.1 Short Answer Template Word Doc.**

- What accuracy do you get with the standard decision tree (no pruning, control of depth)?
    - *To get the plot of accuracy, you can re-use your code from 2.1. Make sure to set the test size to 0.10. Plot the accuracy (train and test) as you vary the depth of the tree from 1 to 15.*
- What does this say about the accuracy of decision trees using text data?
- What accuracy do you get by varying the decision tree depth? Do you get higher accuracy with the entire vocabulary, or does choosing a smaller set of words increase accuracy? What does this suggest about text classification in terms of how much “information” a particular word might be providing the algorithm to make its decision with.

2.3: Audio Data (Drum Sounds): Complete this in **Google Colab**.

- Download the audio_data.csv file (located in your assignment prompt in Blackboard).
- Split the dataset using the following line:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, random_state=42)
```

- Make a plot of accuracy vs depth using these data; you can re-use your code from 2.1. Make sure to set the test size to 0.20. Plot the accuracy (train and test) as you vary the depth of the tree from 1 to 15.
- Set the depth of the tree to 5. Plot the tree (again, you can re-use the code from earlier).
- Delete one training example from the audio training set.
    - `Hint, you can drop examples in pandas`
    - `X_dropped = X.drop(1)`
    - `y needs to be a series for this to work`
    - `y= pd.Series(y)`
    - `y_drop = y.drop(1)`
    - `A better way to do this might be to drop rows from audio_data before you separate the columns out for X and y`
- Re-run your decision tree algorithm. Display "before" and "after" decision trees in your pdf submission.

Answer this question in your **Assignment 2.1 Short Answer Template Word Doc**.

- You made plots of the "before" and "after" decision trees. Refer back to these plots for the following questions:
    - Based on how much the tree changes in these two datasets, what can you say about the stability of decision trees?
    - If two students develop decision tree classifiers using the audio dataset, are their trees likely to be similar or not?

2.4: Decision Trees Summary: Answer these questions in your **Assignment 2.1 Short Answer Template Word Doc**.

- Which dataset has the best accuracy, using decision trees? These aren't exactly apples-to-apples comparisons, but using both your quantitative measures of accuracy and your subjective impression from manipulating the data, which of the three datasets gave the best performance?
- What types of data seem to be best suited for decision tree classification? In this question, be more general than just the three data types - explain what might make any new dataset good or bad for decision tree classification. Answer by explaining in terms of inductive bias.

3. **Cross-Validation:**

In this section, you will look more into cross-validation and model selection. These skills will be useful as you develop more complex models.

Complete #1-3 and 5 in **Google Colab**. Complete #4 and 6-8 in your **Assignment 2.1 Short Answer Template Word Doc**.

1. Plot accuracy vs. depth of the decision tree. Make sure you still have the audio data example loaded. Split the data in this audio dataset using the following line: `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.50, random_state=42)`
2. In the last section of the template, you are given a python class for BadClassifiers. As you can see from the source code, this will make random guesses of labels, but no learning is going on.
3. Re-use the code to train and plot accuracy as tree depth is varied. In this case, every time you increase the depth, get new random predictions, see the assignment code for an example of how to do this.
4. You should have a plot with three lines, showing train accuracy, test accuracy, and random predictions. Does it seem that decision trees outperform random guessing when using the audio dataset?
5. Repeat, using a smaller amount of test data, using the line `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.01, random_state=42)`
6. Plot accuracy vs. depth again, using the smaller amount of training data. How has the plot changed: how different are the training and test set error for the decision tree. How different are the accuracy predictions for random guessing using this smaller test set? Keep in mind that you are making fewer predictions, so random guessing might do well at some times.
7. Why would you use one out cross-validation as opposed to a different method?
8. When would you use a third validation set in addition to using a training and test set?