CM146, Fall 2022
Problem Set 1: Decision trees and k-Nearest Neighbors
Due Oct. 25, 2022 at 11:59 pm

1 Splitting Heuristic for Decision Trees [25 pts]

Recall that the ID3 algorithm iteratively grows a decision tree from the root downwards. On each iteration, the algorithm replaces one leaf node with an internal node that splits the data based on one decision attribute (or feature). In particular, the ID3 algorithm chooses the split that reduces the entropy the most, but there are other choices. For example, since our goal in the end is to have the lowest error, why not instead choose the split that reduces error the most? In this problem, we will explore one reason why reducing entropy is a better criterion.

Consider the following setting. Let us suppose each example is described by 4 boolean features: $X = \langle X_1, \ldots, X_4 \rangle$, where $X_i \in \{0, 1\}$. Furthermore, the target function to be learned is $f : X \rightarrow Y$, where $Y = X_1 \wedge X_2$. That is, $Y = 1$ if $X_1 = 1$ and $X_2 = 1$, otherwise $Y = 0$. Suppose that you have the following training data contains all of 2^4 possible examples:

X_1	X_2	X_3	X_4	Y
0	0	0	0	0
1	0	0	0	0
0	1	0	0	0
1	1	0	0	1
0	0	1	0	0
1	0	1	0	0
0	1	1	0	0
1	1	1	0	1

X_1	X_2	X_3	X_4	Y
0	0	0	1	0
1	0	0	1	0
0	1	0	1	0
1	1	0	1	1
0	0	1	1	0
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

(a) **(5 pts)** Consider constructing a decision tree with only one leaf (the tree in which root is a leaf node and has no internal node) based on these 4 attributes. What is the best 1-leaf decision tree and what is its error rate (i.e., number of mistakes / total number of data) on these 2^4 training examples?

Solution:

The best 1-leaf decision tree is a single node with the label 0. This will predict $Y = 0$ for any given inputs, resulting in a $\frac{4}{16} = 0.25$ error rate.

Parts of this assignment are adapted from course material by Andrea Danyluk (Williams), Tom Mitchell, Matt Gormley and Maria-Florina Balcan (CMU), Stuart Russell (UC Berkeley), Carlos Guestrin (UW), Dan Roth (UPenn) and Jessica Wu (Harvey Mudd).

(b) **(5 pts)** Follow the previous question. Now, let's consider constructing a decision tree with one split. Is there a split that can reduce the error rate? Please specify the attribute that can reduce the error rate if your answer is yes. Otherwise, please discuss why is not.

Solution:

There is no single split that can reduce the error rate. Splitting on X_1, where $X_1 = 0$ predicts $Y = 0$ and $X_1 = 1$ predicts $Y = 1$ still results in a 0.25 error rate. Doing the same with X_2 results in the same error rate, which is the lowest we can achieve with a single split. This is because one split still doesn't create enough partitions such that more than 12 inputs can be mapped to the correct outputs.

(c) **(5 pts)** What is the entropy of the output label $H(Y)$ (rounding to 2 decimal places).

Solution:

$$\begin{aligned}
H(Y) &= -P_+ \log_2(P_+) - P_- \log_2(P_-) \\
&= -\frac{4}{16}\log_2\left(\frac{4}{16}\right) - \frac{12}{16}\log_2\left(\frac{12}{16}\right) \\
&= -\frac{4}{16}(-2) - \frac{12}{16}(-0.415) \\
&= 0.5 + 0.31 \\
&= \boxed{0.81}
\end{aligned}$$

(d) **(5 pts)** What is the information gain if we split the data by the attribute X_1? (rounding to 2 decimal places)

Solution:

$$\begin{aligned}
H(Y_{X_1=0}) &= -P_+ \log_2(P_+) - P_- \log_2(P_-) \\
&= 0 - \frac{8}{8}\log_2\left(\frac{8}{8}\right) \\
&= 0 + 0 \\
&= 0
\end{aligned}$$

$$\begin{aligned}
H(Y_{X_1=1}) &= -P_+ \log_2(P_+) - P_- \log_2(P_-) \\
&= -\frac{4}{8}\log_2\left(\frac{8}{8}\right) - \frac{4}{8}\log_2\left(\frac{8}{8}\right) \\
&= \frac{1}{2} + \frac{1}{2} \\
&= 1
\end{aligned}$$

$$\begin{aligned}
Gain(Y, X_1) &= H(Y) - \left(\frac{|Y_{X_1=0}|}{|Y|}H(Y_{X_1=0}) + \frac{|Y_{X_1=1}|}{|Y|}H(Y_{X_1=1})\right) \\
&= 0.81 - \left(\frac{8}{16}(0) + \frac{8}{16}(1)\right) \\
&= \boxed{0.31}
\end{aligned}$$

(e) **(5 pts)** What is the information gain if we split the data by the attribute X_3? (rounding to 2 decimal places)

Solution:

$$\begin{aligned} H(Y_{X_3=0}) &= -P_+ \log_2(P_+) - P_- \log_2(P_-) \\ &= \frac{2}{8}\log_2\left(\frac{2}{8}\right) - \frac{6}{8}\log_2\left(\frac{6}{8}\right) \\ &= -\frac{1}{4}(-2) - \frac{3}{4}(-0.415) \\ &= 0.81 \end{aligned}$$

$$\begin{aligned} H(Y_{X_3=1}) &= -P_+ \log_2(P_+) - P_- \log_2(P_-) \\ &= \frac{2}{8}\log_2\left(\frac{2}{8}\right) - \frac{6}{8}\log_2\left(\frac{6}{8}\right) \\ &= -\frac{1}{4}(-2) - \frac{3}{4}(-0.415) \\ &= 0.81 \end{aligned}$$

$$\begin{aligned} Gain(Y, X_3) &= H(Y) - \left(\frac{|Y_{X_3=0}|}{|Y|} H(Y_{X_3=0}) + \frac{|Y_{X_3=1}|}{|Y|} H(Y_{X_3=1})\right) \\ &= 0.81 - \left(\frac{8}{16}(0.81) + \frac{8}{16}(0.81)\right) \\ &= \boxed{0} \end{aligned}$$

2 k-Nearest Neighbors and Cross-validation [20 pts]

In the following questions you will consider a k-nearest neighbor classifier using the Euclidean distance metric on a binary classification task. We assign the class of the test data point to be the class of the majority of the k nearest neighbors. Note that when the test data point is the same as one of the training data point. That training data point can be consider as the closet neighbor of the test data point.

(a) **(5 pts)** What will be the label of point (5,9) in Fig 1 using k-NN algorithm with majority voting when $k = 1$?

Solution:

$\boxed{+}$

(b) **(5 pts)** What will be the label of point (5,9) in Fig 1 using k-NN algorithm with majority voting when $k = 3$?

Solution:

$\boxed{-}$

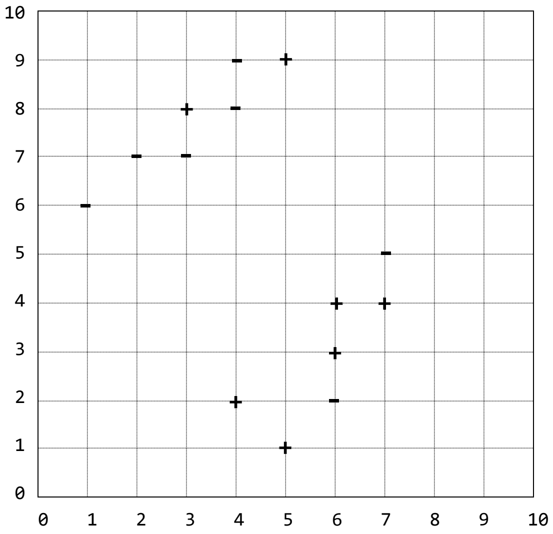
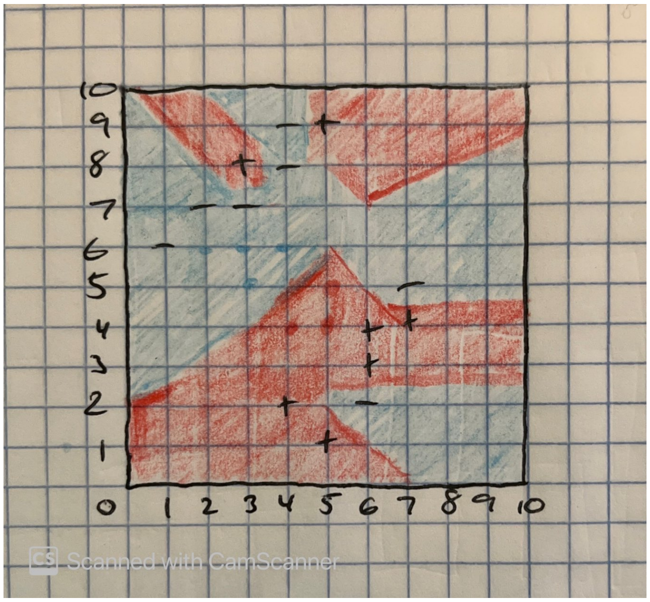


Figure 1: Dataset for KNN binary classification task.

(c) **(10 pts)** Draw the decision boundary of k-NN when $k = 1$ on Fig 1.

Solution:

Red = +, Blue = -



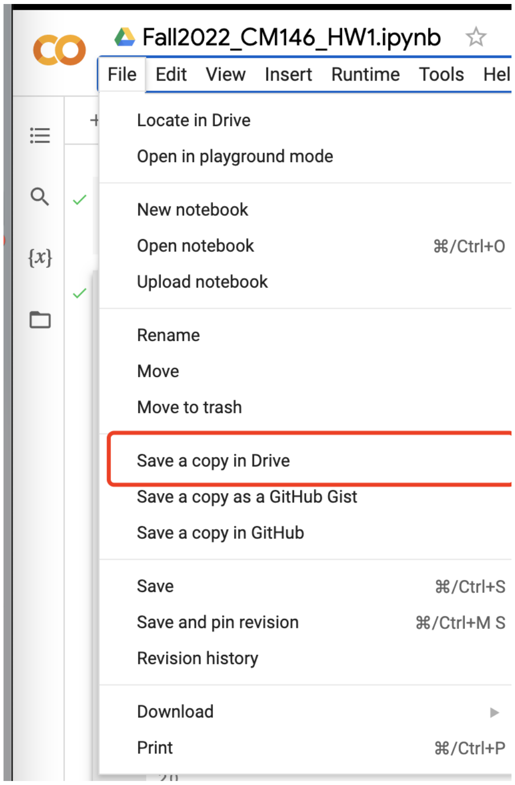
3 Programming exercise : Applying decision trees and k-nearest neighbors [55 pts]

Introduction

In this problem, we will work on a mushroom classification task. The dataset is adapted from the UCI Machine Learning Repository and it contains descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms. Each mushroom is described in terms of physical characteristics, and the goal is to classify mushrooms as *edible* or *poisonous*. We will apply decision trees and k-nearest neighbors. Since this dataset is relatively simple for classification, we only use 6 features out of 22 features in the original dataset. Features we use include: cap-shape, cap-color, gill-color, stalk-root, veil-type, ring-number.

For all the coding, please refer to the following Colab notebook Fall2022-CM146-HW1.ipynb.

Before executing or writing down any code, please make a copy of the notebook and save it to your own google drive by clicking the "File" → "Save a copy in Drive".



You will probably be prompted to log into your Google account. Please make sure all the work you implement is done on your own saved copy. You won't to able to make changes on the the original notebook shared with the entire class.

The notebook has marked blocks where you need to code:

```
### ========== TODO : START ========== ###
### ========== TODO : END ========== ###
```

Submission instructions for programming problems

- Please save the execution output in your notebook. When submitting, please export the notebook to a `.ipynb` file by clicking "File" → "Download .ipynb" and upload the notebook to BruinLearn.
- Your code should be commented appropriately. Importantly:
  - Your name should be at the top of the file.
  - Each class and method should have an appropriate doctsring.
  - Include some comments for anything complicated.

  There are many possible solutions to this assignment, which makes coding style and comments important for graders to conveniently understand the code.
- Please submit all the plots and the rest of the solutions (other than codes) to Gradescope.

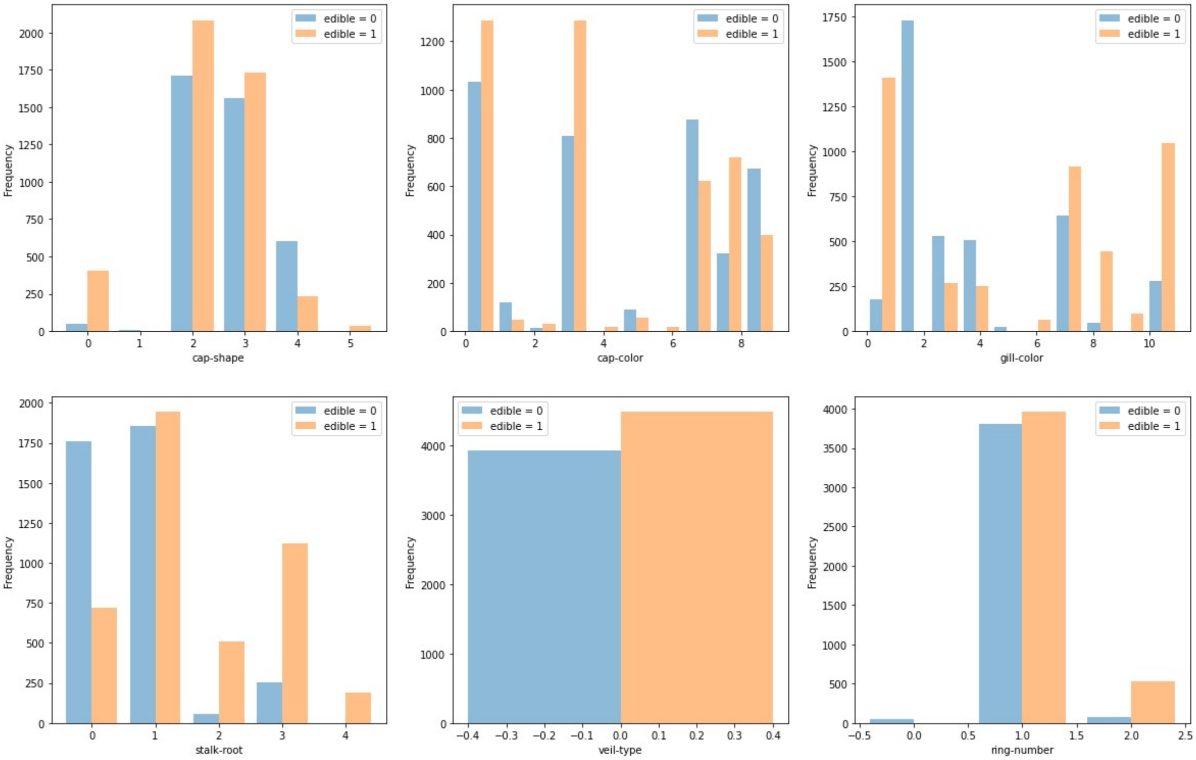
3.1 Visualizing Features [5 pts]

One of the first things to do before trying any formal machine learning technique is to dive into the data. This can include looking for funny values in the data, looking for outliers, looking at the range of feature values, what features seem important, etc. We have already included the code for loading the data, converting all the categorical features to numerical one.

Make histograms for each feature, separating the examples by class (i.e., edible or poisonous). This should produce 6 plots, one for each feature, and each plot should have two overlapping histograms, with the color of the histogram indicating the class. The code has been included in `plot_histograms` and `plot_histogram` functions, and you do not need to code by yourself.

For each feature, what trends do you observe in the data? (Please only describe the general trend. No need for more than two sentences per feature.)

Solution:



The cap-shape data appears to show that mushrooms of cap-shape 2 and 3 are more likely to be edible, but those with a cap-shape value of 4 are more likely to be inedible. Other values of cap-shape have significantly less data, and may not provide any meaningful insights due to the small sample size.

The cap-color data shows that mushrooms of cap-color 0, 3, and 8 are more likely to be edible, while mushrooms of cap-color 7 and 9 are more likely to be inedible. Other values of cap-color have significantly less data, and may not provide any meaningful insights due to the small sample size.

The gill-color data shows that mushrooms with lower values of gill-color (excluding 0) are more likely to be inedible, while mushrooms with higher values of gill-color or a gill-color value of 0 are more likely to be edible.

The stalk-root data shows that mushrooms with stalk-root value of 0 are more likely to be inedible, mushrooms with stalk-root value of 1 are about equally likely to be edible or inedible, and mushrooms with stalk-root values of 2 or above are likely to be edible.

The veil-type data shows that veil-type is likely not a useful characteristic of mushrooms in determining their edibility, as there is a relatively even split between edible and inedible based on this variable.

The ring-number data shows that ring-number is likely not a useful characteristic of mushrooms in determining their edibility, as there is a relatively even split between edible and inedible based on values of this variable.

3.2 Training and Evaluating Models [55 pts]

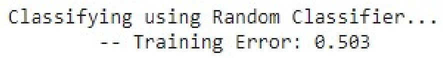
Now, let's use scikit-learn to train a DecisionTreeClassifier and KNeighborsClassifier on the data. Using the predictive capabilities of the scikit-learn package can be carried out in three steps: initializing the model, fitting it to the training data, and predicting new values.

(a) **(5 pts)** Before trying out any classifier, it is often useful to establish a *baseline*. We have implemented one simple baseline classifier, MajorityVoteClassifier, that always predicts the majority class from the training set. Read through the MajorityVoteClassifier and its usage and make sure you understand how it works.

Your goal is to implement and evaluate another baseline classifier, RandomClassifier, that predicts a target class according to the distribution of classes in the training data set. For example, if 85% of the examples in the training set have edible = 0 and 15% have edible = 1, then, when applied to a test set, RandomClassifier should randomly predict 85% of the examples as edible = 0 and 15% as edible = 1.

Implement the missing portions of RandomClassifier according to the provided specifications. Then train your RandomClassifier on the entire training data set, and evaluate its training error.

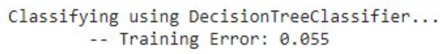
Solution:



```
Classifying using Random Classifier...
        -- Training Error: 0.503
```

(b) **(5 pts)** Now that we have a baseline, train and evaluate a DecisionTreeClassifier (using the class from scikit-learn and referring to the documentation as needed). Make sure you initialize your classifier with the appropriate parameters; in particular, use the 'entropy' criterion discussed in class. What is the training error of this classifier?

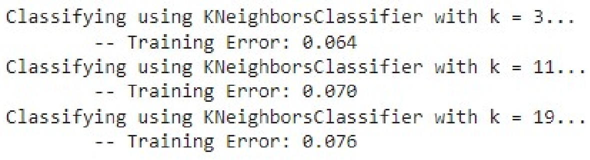
Solution:



```
Classifying using DecisionTreeClassifier...
        -- Training Error: 0.055
```

(c) **(10 pts)** Similar to the previous question, train and evaluate a `KNeighborsClassifier` (using the class from `scikit-learn` and referring to the documentation as needed). Use k=3, 11 and 19 as the number of neighbors and report the training error of this classifier. If we implement KNN model from scratch, what operations we should do in the `fit` method?

Solution:



```
Classifying using KNeighborsClassifier with k = 3...
        -- Training Error: 0.064
Classifying using KNeighborsClassifier with k = 11...
        -- Training Error: 0.070
Classifying using KNeighborsClassifier with k = 19...
        -- Training Error: 0.076
```

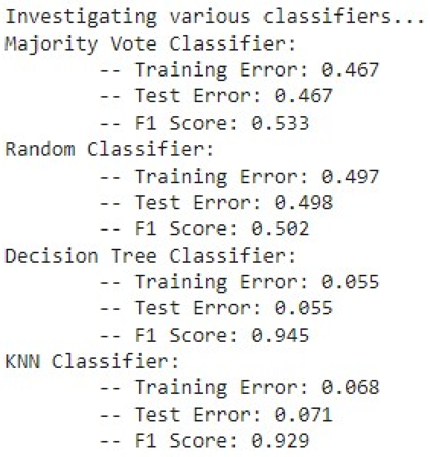
If we were to implement the KNN model from scratch, we would want the `fit` method to associate each feature vector in the training set with its corresponding label within a data structure. This would allow us to use this data structure while predicting to find the k nearest neighbors to a given test point, and to easily access their labels and generate decision boundaries.

(d) **(10 pts)** So far, we have looked only at training error, but as we learned in class, training error is a poor metric for evaluating classifiers. Let's use cross-validation instead.

Implement the missing portions of `error(...)` according to the provided specifications. You may find it helpful to use `StratifiedShuffleSplit(...)` from `scikit-learn`. To ensure that we always get the same splits across different runs (and thus can compare the classifier results), set the `random_state` parameter to be the same (e.g., 0).

Next, use your `error(...)` function to evaluate the training error and (cross-validation) test error and test micro averaged F1 Score of each of your four models (for the `KNeighborsClassifier`, use $k = 11$). To do this, generate a random 85/15 split of the training data, train each model on the 85% fraction, evaluate the error on both the 85% and the 15% fraction, and repeat this 100 times to get an average result.

Solution:



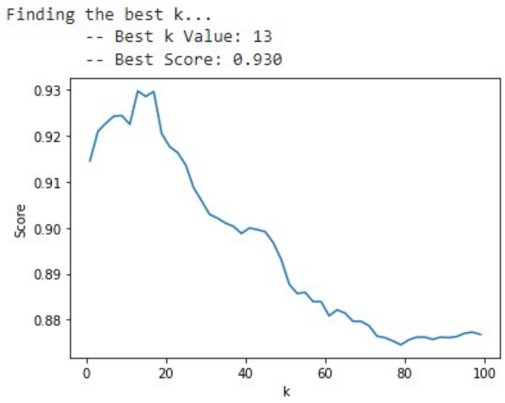
```
Investigating various classifiers...
Majority Vote Classifier:
        -- Training Error: 0.467
        -- Test Error: 0.467
        -- F1 Score: 0.533
Random Classifier:
        -- Training Error: 0.497
        -- Test Error: 0.498
        -- F1 Score: 0.502
Decision Tree Classifier:
        -- Training Error: 0.055
        -- Test Error: 0.055
        -- F1 Score: 0.945
KNN Classifier:
        -- Training Error: 0.068
        -- Test Error: 0.071
        -- F1 Score: 0.929
```

(e) **(10 pts)** One way to find out the best value of k for `KNeighborsClassifier` is n-fold cross validation. Find out the best value of k using 5-fold cross validation and the F1 Score metric. Run cross validation for all odd numbers ranging from 1 to 100 as the number of neighbors.

Then plot the validation score against the number of neighbors, k. Include this plot in your writeup. What is the best value of k and what is the corresponding score?

You may find the `cross_val_score(...)` from `scikit-learn` helpful.

Solution:



```
Finding the best k...
        -- Best k Value: 13
        -- Best Score: 0.930
```

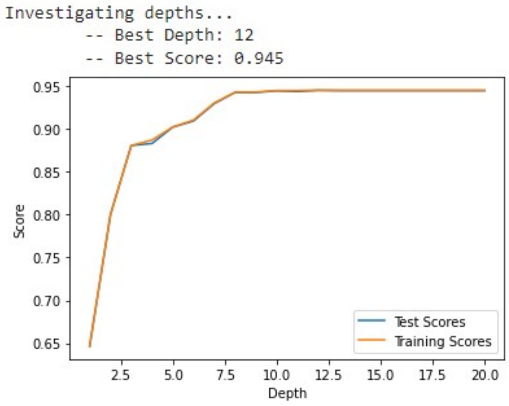
(f) **(10 pts)** One problem with decision trees is that they can *overfit* to training data, yielding complex classifiers that do not generalize well to new data. Let's see whether this is the case.

One way to prevent decision trees from overfitting is to limit their depth. Run 20-fold cross-validation for increasing depth limits $1, 2, \ldots, 20$.

Then plot the average training F1 Score and test F1 score against the depth limit. Include this plot in your writeup, making sure to label all axes and include a legend for your classifiers. What is the best depth limit to use for this data? Do you see overfitting? Justify your answers using the plot.

You may find `cross_validate` from `scikit-learn` helpful when you want both training scores and validation scores.

Solution:



```
Investigating depths...
        -- Best Depth: 12
        -- Best Score: 0.945
```

We do see evidence of overfitting in this case. Initially, as we increase the depth, our average validation score increases as well. This is a result of underfitting, where our decision tree is not deep enough to take advantage of the data available to us. However, once we pass a depth of 12, we see a decrease in the average validation score, telling us that trees of depth greater than 13 actually overfit the training data, resulting in worse F1 scores.