Machine Learning - Homework 1

Author: Jordy A. Larrea Rodriguez

```
import numpy as np
%load_ext autoreload
%autoreload 2
```

Function Definitions

```
# Returns the positive proportion of labels. p = # positive labels / # labels
def get_binary_p(S: dict):

    labels = np.array(list(S.values()))
    return labels.sum()/len(labels)

# Returns the dataset Entropy given the proportion of positive labels for a binary classif
def get_entropy(p):
    eps = 1e-10
    return -p*np.log2(p) - (1-p + eps)*np.log2(1-p + eps)

# Returns the information gain for an attribute 'a_i'.
def get_info_gain(S, a_i, entropy_S, attribute_counts, plus_counts):
    S_length = len(S)

    Sv_lengths = np.array([attribute_counts[v] for v in a_i])
    p_counts = np.array([plus_counts[v] for v in a_i])

    length_ratios = Sv_lengths / (np.ones_like(Sv_lengths)*S_length)
    Sv_entropies = get_entropy(p_counts/Sv_lengths)
    return entropy_S - np.sum(length_ratios*Sv_entropies)

# Calculates and returns the information gains (Gain(S,A)) for all attributes a_i in A in
def gain(S, A):
```

```
    plus_counts = {}; attribute_counts = {}
    for feature, label in S.items():
        for v in feature:
            plus_counts[v] = plus_counts.get(v, 0) + int(label)
            attribute_counts[v] = attribute_counts.get(v, 0) + 1

    H_S = get_entropy(get_binary_p(S))

    return [get_info_gain(S, a_i, H_S, attribute_counts, plus_counts) for a_i in A]
```

Part 1 - Decision Trees

Q1

Dataset S:

Variety	*Color*	*Smell*	*Time*	*Ripe?*
Alphonso	*Red*	*None*	*Two*	*False*
Keitt	*Red*	*None*	*One*	*True*
Alphonso	*Yellow*	*Sweet*	*Two*	*True*
Keitt	*Green*	*None*	*Two*	*False*
Haden	*Green*	*Sweet*	*One*	*True*
Alphonso	*Yellow*	*None*	*Two*	*False*
Keitt	*Yellow*	*Sweet*	*One*	*False*
Alphonso	*Red*	*Sweet*	*Two*	*True*

```
S = {
    ('A', 'R', 'N', 'T') : False,
    ('K', 'R', 'N', 'O') : True,
    ('A', 'Y', 'S', 'T') : True,
    ('K', 'G', 'N', 'T') : False,
    ('H', 'G', 'S', 'O') : True,
    ('A', 'Y', 'N', 'T') : False,
    ('K', 'Y', 'S', 'O') : False,
    ('A', 'R', 'S', 'T') : True,
}

variety = {'A', 'K', 'H'}; color = {'R', 'Y', 'G'}; smell = {'N', 'S'}; time = {'O', 'T'}

A = [variety, color, smell, time]; A_names = ["Variety", "Color", "Smell", "Time"]
```

1.a.1 - How many possible functions are there to map these four features to a boolean decision?

Response: If we consider the feature space to consist of six features, $A = \{ \vec{a_1}\ \vec{a_2}\ \vec{a_3}\ \vec{a_4}\ \vec{a_5}\ \vec{a_6}\ \}$ where $\{\ a_1\ a_2\ \}$ enumerate the Variety feature and $\{\ a_3\ a_4\ \}$ enumerates the Color feature. The entire table for the aforementioned set of features would then result in $2^6 = 64$ rows; thereby, having 2^{64} possible solution functions. If we restrict the hypothesis space to only consider possible enumerations, then the resulting table will have $3 * 3 * 2 * 2 = 36$ rows, and 2^{36} possible functions.

1.a.2 - How many functions are consistent with the given training dataset?

Response: The dataset has 8 rows; thus, only $2^8 = 256$ functions are consistent w/ the dataset.

1.b - How many functions are consistent with the given training dataset?

Entropy:

$$H(S) = -p\, log_2(p)\ -\ (1-p)\, log_2(1-p), \text{ where p is the probability of a success.}$$

$$p\ =\ 0.5$$

```
# Here p is the probability of + in the dataset and H_S is the entropy of dataset S.
print(f"The proportion of positive labels is {get_binary_p(S)}")
print(f"The entropy of dataset S is {get_entropy(get_binary_p(S))}.")
```

```
The proportion of positive labels is 0.5
The entropy of dataset S is 0.9999999999557305.
```

1.c - Compute the information gain of each feature.

Information Gain for a dataset S with attribute/feature $\vec{a_i} \in A$:

$Gain(S, \vec{a_i})\ =\ H(S) - \Sigma_{v \in \vec{a_i}} \frac{\|S_v\|}{\|S\|} H(S_v)$, where S_v is the subset of S containing attribute value v.

```
print("Dataset S: \n\nAttributes\t\t | Ripe?")
print("_"*45)
for s in S.items(): print(f"{s[0]}\t | {s[1]}")

print("\nInformation Gain Table:\n")
print(f"A\t | Information Gain")
```

```
print("_"*45)

gain_S = gain(S, A)
for i, g_i in enumerate(gain_S):
    print(f"{A_names[i]}\t | {g_i}")
```

```
Dataset S:

Attributes       | Ripe?
---------------------------------------------
('A', 'R', 'N', 'T')     | False
('K', 'R', 'N', 'O')     | True
('A', 'Y', 'S', 'T')     | True
('K', 'G', 'N', 'T')     | False
('H', 'G', 'S', 'O')     | True
('A', 'Y', 'N', 'T')     | False
('K', 'Y', 'S', 'O')     | False
('A', 'R', 'S', 'T')     | True

Information Gain Table:

A    | Information Gain
---------------------------------------------
Variety  | 0.1556390618243556
Color    | 0.06127812445276071
Smell    | 0.18872187552011532
Time     | 0.0487949406899022
```

1.d - Which attribute will you use to construct the root of the tree using the information gain heuristic of the ID3 algorithm?

Response: I would use the "Smell" attribute since the attribute maximizes the gain which minimizes the entropy or disorder in the dataset.

1.e - Using the root that you selected in the previous question, construct a decision tree that represents the data. You do not have to use the ID3 algorithm here, you can show any tree with the chosen root.

Note : for some reason my image is not showing up on the printout. Please refer to the hw1/img/ folder.

1.f - Suppose you are given three more examples. Use your decision tree to predict the label for each example. Also report the accuracy of the classifier that you have learned.

Three New Examples:

Variety	*Color*	*Smell*	*Time*	*Ripe?*
Alphonso	*Green*	*Sweet*	*Two*	*True*
Keitt	*Red*	*Sweet*	*One*	*False*
Haden	*Yellow*	*None*	*Two*	*True*

Response: My decision is quite simple given the heuristics, and fits well to the new examples. Because the first two examples have a "sweet" smell, they are reduced to a binary decision where the Alphonso variety results in a ripe fruit while the Keitt variety does not. My decision tree assumes that a Haden mango that lacks a smell always results in an unripe fruit which aligns with my decision tree. Thus, my tree has a 100% classification accuracy given the three new examples.

Q2 - ID3 Algorithm

Recall that in the ID3 algorithm, we want to identify the best attribute that splits the examples that are relatively pure in one label. Aside from entropy, which we saw in class and you used in the previous question, there are other methods to measure impurity. We will now develop a variant of the ID3 algorithm that does not use entropy. If, at some node, we stopped growing the tree and assign the most common label of the remaining examples at that node, then the empirical error on the training set S at that node will be

$ME(S) = 1 - \max_i p_i$, where p_i is the fraction of examples that are labeled with the i^{th} label.

Furthermore, ME can be thought as the minimum proportion for the binary labels. That is...

$ME(S) = min(p_+, p_-) = min(p, 1-p)$, where p is the fraction of examples that are + labeled.

2.a - Notice that $MajorityError$ can be thought of as a measure of impurity just like entropy. Just like we used entropy to define information gain, we can define a new version of information gain that uses $MajorityError$ (ME) in place of entropy. Write down an expression that defines a new version of information gain that uses $MajorityError$ in place of entropy.

Information Gain for a dataset S with attribute/feature $\vec{a_i} \in A$:

$Gain_{ME}(S, \vec{a_i}) = ME(S) - \Sigma_{v \in \vec{a_i}} \frac{\|S_v\|}{\|S\|} ME(S_v)$, where S_v is the subset of S containing attribute value v.

2.b - Calculate the value of your newly defined information gain from the previous question for the four features in the mango dataset.

$$ME = min(p, 1-p) = \frac{1}{2}$$

Feature	Information Gain (using majority error)
Variety	$G(S, variety) = \frac{1}{2} - (\frac{1}{2}(\frac{1}{2}) + \frac{3}{8}(\frac{1}{3}) + \frac{1}{8}(0)) = \frac{1}{8}$
Color	$G(S, variety) = \frac{1}{2} - (\frac{3}{8}(\frac{1}{3}) + \frac{3}{8}(\frac{1}{3}) + \frac{1}{4}(\frac{1}{2})) = \frac{1}{8}$
Smell	$G(S, variety) = \frac{1}{2} - (\frac{1}{2}(\frac{1}{4}) + \frac{1}{2}(\frac{1}{4})) = \frac{1}{4}$
Time	$G(S, variety) = \frac{1}{2} - (\frac{3}{8}(\frac{1}{3}) + \frac{5}{8}(\frac{2}{5})) = \frac{1}{8}$

2.c - According to your results in the last question, which attribute should be the root for the decision tree? Do these two measures (entropy and majority error) lead to the same tree?

Response: The root node should be "Smell" since it maximizes the gain; however, because the other attributes produce the same gain of $\frac{1}{8}$, the resulting tree would depend on the tie-breaking strategy employed. Therefore, the ME heuristic might lead to the same tree, but it is not guaranteed.

Part 2 - Experiments

```
# Part 2 Setup
import math
import numpy as np
import pandas as pd
from utils import load_data, get_attribute_dict, calc_acc
from decision_tree import DecisionTree

# load data
labels = {True:'p', False:'e'}

x_train, y_train, data_train = load_data(list(labels.values()), null='?', dir=r'data/train
x_test, y_test, data_test = load_data(list(labels.values()), null='?', dir=r'data/test.csv

attribute_dict = get_attribute_dict(pd.concat([x_train, x_test]), [('veil-type','u')])
```

Q1 - Baseline

1.a - First, find the most common label in the training data. What is the training and test accuracy of a classifier that always predicts this label?

```
p_count_train = len(np.where(y_train)[0]); e_count_train = len(np.where(~y_train)[0])
p_count_test = len(np.where(y_test)[0]); e_count_test = len(np.where(~y_test)[0])
print(f"The 'poisonous' and 'edible' labels occur {p_count_train} and {e_count_train} time
print(f"The most common label in this training data is '{labels[p_count_train > e_count_tr
print(f"\nA classifier that always predicts 'e' gets an accuracy of {round(e_count_train/(
print(f"A classifier that always predicts 'e' gets an accuracy of {round(e_count_test/(e_c
```

```
The 'poisonous' and 'edible' labels occur 3148 and 3382 times respectively
The most common label in this training data is 'e'.

A classifier that always predicts 'e' gets an accuracy of 51.792% on the train set.
A classifier that always predicts 'e' gets an accuracy of 51.819% on the test set.
```

Q2 - Full Tree Implementation

In the first set of decision tree experiments, run the ID3 algorithm we saw in class without any depth restrictions. (That is, there are no hyperparameters for this setting.) [6 points] Implement the decision tree data structure and the ID3 algorithm for your decision tree (Remember that the decision tree need not be a binary tree!).

```
# Test Tree for Q2 on Provided Datasets
q2_test = DecisionTree(a_dict=attribute_dict)
q2_test.train(x_train, y_train, labels)
```

2.a - The root feature that is selected by your algorithm.

```
print(f"The root attribute selected by my algorithm is: '{q2_test.get_max_gain()[0]}'.")
```

```
The root attribute selected by my algorithm is: 'spore-print-color'.
```

2.b - Information gain for the root feature.

```
print(f"The information gain of the root feature is: {round(q2_test.get_max_gain()[1], 3)}
```

```
The information gain of the root feature is: 0.485.
```

2.c - Maximum depth of the tree that your implementation gives.

```
print(f"The height of my tree via DFS traversal: {q2_test.depth}")
print(f"The height of my tree via BFS traversal: {q2_test.calculate_tree_depth()}")
```

```
The height of my tree via DFS traversal: 12
The height of my tree via BFS traversal: 12
```

2.d - Accuracy on the training set.

```
train_pred = q2_test.predict(x_train)
print(f"The prediction accuracy of my classifier on the training set is {round(calc_acc(tr
```

```
The prediction accuracy of my classifier on the training set is 100.0%.
```

2.e - Accuracy on the test set.

```
test_pred = q2_test.predict(x_test)
print(f"The prediction accuracy of my classifier on the testing set is {round(calc_acc(tes
```

```
The prediction accuracy of my classifier on the testing set is 100.0%.
```

Q3 - Limiting Depth

Next, you will perform 5-fold cross-validation to limit the depth of your decision tree, effectively pruning the tree to avoid overfitting. We have already randomly split the training data into five splits. You should use the 5 cross-validation files for this section, titled data/CVfolds/foldX.csv where X is a number between 1 and 5 (inclusive).

3.a - Run 5-fold cross-validation using the specified files. Experiment with depths in the set {1, 2, 3, 4, 5, 10, 15}, reporting the average cross-validation accuracy and standard deviation for each depth. Explicitly specify which depth should be chosen as the best, and explain why. If a certain depth is not feasible for any reason, your report should explain why.

```
from utils import load_data, calc_acc
from decision_tree import DecisionTree
import numpy as np
import pandas as pd
# Set for Depth Hyperparameter
max_depths = {1, 2, 3, 4, 5, 10, 15}
labels = {True:'p', False:'e'}
```

```
# Load K-fold datasets
K = 5
k_datasets = [load_data(list(labels.values()), null='?', dir=f'./data/CVfolds_new/fold{k}.
dataset = pd.concat(k_datasets)
attribute_dict = get_attribute_dict(dataset.loc[:, ~dataset.columns.isin(['label'])], [('v
```

Five-Fold Cross-Validation Experiment

```
cv_acc = {md : [] for md in max_depths}

# 5-fold cross-validation
for md in max_depths:
    for k in range(K):
        k_ds = list(k_datasets)

        # Derive validation set
        data_val = k_ds[k]
        x_val = data_val.loc[:, ~data_val.columns.isin(['label'])]
        y_val = (data_val.loc[:, data_val.columns.isin(['label'])].to_numpy() == list(labe

        k_ds.pop(k)

        # Derive training set
        data_train = pd.concat(k_ds)
        x_train = data_train.loc[:, ~data_train.columns.isin(['label'])]
        y_train = (data_train.loc[:, data_train.columns.isin(['label'])].to_numpy() == lis


        dt = DecisionTree(a_dict=attribute_dict)
        dt.train(x_train, y_train, labels, max_height=md)

        cv_acc[md].append(calc_acc(dt.predict(x_val), y_val))

print("Results from five-fold cross-validation:\n")
print("depth 'd'\t| five-fold cross-validation Accuracies")
for md, trials in cv_acc.items():
    print(f"d={md}\t\t| {np.round(trials,3)}")
```

```
Results from five-fold cross-validation:

depth 'd'   | five-fold cross-validation Accuracies
```

```
d=1     | [0.116 0.112 0.335 0.169 0.312]
d=2     | [0.116 0.117 0.335 0.169 0.312]
d=3     | [0.116 0.132 0.547 0.685 0.312]
d=4     | [0.152 0.304 0.547 0.828 0.765]
d=5     | [0.162 0.305 0.771 0.828 0.75 ]
d=10        | [0.946 1.    0.985 0.974 0.975]
d=15        | [0.949 1.    0.985 0.974 0.975]
```

```python
# Take Average across Five-Fold Split Trials

cv_acc_stats = {md : (np.mean(trials), np.std(trials)) for md, trials in cv_acc.items()}

print("Accuracy statistics from five-fold cross-validation:\n")
print("depth 'd' \t| five-fold mean | five-fold std")
for md, (mean, std) in cv_acc_stats.items():
    print(f"d={md}\t\t| {np.round(mean, 3)}\t\t | {np.round(std, 3)}")
```

```
Accuracy statistics from five-fold cross-validation:

depth 'd'   | five-fold mean | five-fold std
d=1     | 0.209      | 0.096
d=2     | 0.21       | 0.095
d=3     | 0.358      | 0.226
d=4     | 0.519      | 0.26
d=5     | 0.563      | 0.274
d=10        | 0.976      | 0.018
d=15        | 0.977      | 0.016
```

Discussion: My classifier's maximum depth was d=12; thus, depth=15 was not possible for my implementation; therefore, d=15 is actually d=12, thereby, implying that the depth that maximizes the five-fold cross-validation accuracy is d=12 given the improved statistics. The depth, d=10, would also work reasonably well given how closely it matches the statistics in d=12.

3.b - Explain briefly how you implemented the depth limit functionality in your code.

Discussion: Children at the maximum depth were set to the common label if the leafs were nodes that represented attributes; otherwise, the tree was created per-usual.

3.c - Using the depth with the greatest cross-validation accuracy from your experiments: train your decision tree on the data/train.csv file. Report the accuracy of your decision tree on the data/test.csv file.

Response: The ID3 algorithm implementation I wrote produced a tree of depth d=12. I had already experimented w/ the train and test datasets (Part 2 - Q2). The accuracy of my implementation at the optimal depth is 100.0% for both.

3.e - 3.f

Discussion: My depth limited tree was the full decision tree. However, it is clear that trees at depth d=10 could perform reasonably well. Clearly, a smaller tree could substantially reduce the computational demands during training and during prediction.