STAT 760 Final Project

Decision Trees with Varying Maximum Depth on a Stellar Classification Dataset

By: Alexia Spoon

The following dataset uses numerous features to classify an image from a telescope taken by the SDSS (Sloan Digital Sky Survey) as a galaxy, a star, or a quasar object. There are 100,000 observations in this dataset with 16 features. The features being used include: `alpha`, the Right Ascension angle; `delta`, the Declination angle; `u`, the Ultraviolet filter in the photometric system; `g`, the Green filter in the photometric system; `r`, the Red filter in the photometric system; `i`, the Near Infrared filter in the photometric system; `z`, the Infrared filter in the photometric system; `run_ID`, the Run Number used to identify the specific scan; `rerun_ID`, the Rerun Number to specify how the image was processed; `cam_col`, the Camera column to identify the scanline within the run; `field_ID`, the Field number to identify each field; `spec_obj_ID`, the ID used for optical spectroscopic objects; `redshift`, the redshift value based on the increase in wavelength; `plate`, the plate ID; `MJD`, the Modified Julian Date, used to indicate when a given piece of SDSS data was taken; and `fiber_ID`, the fiber ID that identifies the fiber that pointed the light at the focal plane in each observation. The outcome is found in the variable `class`, which has three options: GALAXY, STAR, or QSO. Each feature will be used in a decision tree in order to predict the class of each observation. This is a complex dataset because there are three possible outcomes rather than two, as we have worked with for a majority of the course, as well as the fact that the dataset is immensely large with 100,000 observations.

```
import numpy as np
import pandas as pd
import random as rand
from random import choices
```

```
data = pd.read_csv("/content/star_classification.csv")
```

Since each observation has a unique `obj_ID`, this variable is dropped.

```
data = data.drop(['obj_ID'], axis = 1)
```

The `class` column containing the outcome is recoded as 0 for GALAXY, 1 for QSO, and 2 for STAR to be used in the decision tree.

```
data['class'].replace(['GALAXY', 'QSO', 'STAR'], [0, 1, 2], inplace=True)
```

The training dataset is a random sample of 2/3 of the 100,000 observations, with the test dataset containing the remaining 1/3 of the observations.

```
shuffle_data = data.sample(frac=1)

train_size = int(2/3 * len(data))

data_train = shuffle_data[:train_size]
data_test = shuffle_data[train_size:]


X_train = data_train.drop(['class'], axis = 1)


y_train = data_train['class']


X_test = data_test.drop(['class'], axis = 1)


y_test = data_test['class']
```

The decision tree is produced using the `tree` function created below. This function has 4 parameters: `X_train`, `y_train`, `current_depth`, and `max_depth`. The parameter `X_train` contains the dataset of all the training data and its features. The parameter `y_train` contains the outcomes of the training dataset. The parameter `current_depth` is the current depth of the tree, and the parameter `max_depth` is the maximum depth of the tree. These two parameters are necessary due to the massive size of the dataset. Due to its size and complexity, a maximum depth must be introduced as to limit the size of the tree, for both time and processing power purposes. The maximum depth of the tree will be changed, and each tree's accuracy observed in order to determine the optimal depth of the tree which yields the highest accuracy.

```
def tree(X_train, y_train, current_depth, max_depth):
  if len(X_train) < 50 or current_depth == max_depth:
    return {'prediction': np.mean(y_train)}
  else:
    optimal_col = None
    optimal_split = None
    min_rss = []
    split = []
    col = []
```

```
    for i in X_train.columns:
      splits = X_train[i].unique()
      t_rss = []
      for j in splits.tolist():
        y_left = y_train[X_train[i] < j]
        y_right = y_train[X_train[i] >= j]
        t_rss.append(np.sum((y_left - np.mean(y_left)) ** 2)
          + np.sum((y_right - np.mean(y_right)) ** 2))
      min_rss.append(min(t_rss))
      split.append(splits[np.argmin(t_rss)])
      col.append(i)
    optimal_split = split[np.argmin(min_rss)]
    optimal_col = col[np.argmin(min_rss)]

    left = X_train[optimal_col] < optimal_split
    right = X_train[optimal_col] >= optimal_split
    rule = {'column': optimal_col, 'split': optimal_split}
    rule['left'] = tree(X_train[left], y_train[left], current_depth + 1, max_depth)
    rule['right'] = tree(X_train[right], y_train[right], current_depth + 1, max_depth)
  return rule
```

The `predict` function has two parameters: `sample` and `rules`. The `sample` is a single observation in the test dataset, and the `rules` parameter is the tree previously created. The `sample` will be passed through the `rules` of a given decision tree, and a prediction will be made based on how the observation traverses the decision tree.

```
  def predict(sample, rules):
    prediction = None
    while prediction is None:
      rules = rules['left'] if sample[rules['column']] < rules['split'] else rules['right']
      prediction = rules.get('prediction',None)
    return prediction
```

Now, decision trees with maximum depths of 2, 4, 6, 8, 10, and 12, respectively, are created below using the training dataset. The current depth of each decision tree is set as 0, meaning the first split will take the decision tree to depth 1.

```
rules_depth2 = tree(X_train, y_train, 0, 2)
```

```
rules_depth4 = tree(X_train, y_train, 0, 4)
```

```
rules_depth6 = tree(X_train, y_train, 0, 6)
```

```
rules_depth8 = tree(X_train, y_train, 0, 8)
```

```
rules_depth10 = tree(X_train, y_train, 0, 10)
```

```
rules_depth12 = tree(X_train, y_train, 0, 12)
```

Now, for each row in the test dataset, the decision trees with maximum depths 2, 4, 6, 8, 10, and 12, respectively, are used to predict the outcomes of each observation. Since the predictions for the decision tree is just the mean of the remaining observation's outcomes, the predicted class will be 0 (GALAXY) if the final prediction from the decision tree is less than 0.5, 1 (QSO) if the final prediction from the decision tree is between 0.5 and 1.5, and 2 (STAR) if the final prediction from the decision tree is greater than 1.5.

```
preds2 = []
for index, row in X_test.iterrows():
    preds2.append(predict(row, rules_depth2))

predictions2 = []
for i in preds2:
    if i < 0.5:
        predictions2.append(0)
    elif i >= 0.5 and i < 1.5:
        predictions2.append(1)
    else:
        predictions2.append(2)


preds4 = []
for index, row in X_test.iterrows():
    preds4.append(predict(row, rules_depth4))

predictions4 = []
for i in preds4:
    if i < 0.5:
        predictions4.append(0)
    elif i >= 0.5 and i < 1.5:
        predictions4.append(1)
    else:
        predictions4.append(2)


preds6 = []
for index, row in X_test.iterrows():
    preds6.append(predict(row, rules_depth6))

predictions6 = []
for i in preds6:
    if i < 0.5:
        predictions6.append(0)
    elif i >= 0.5 and i < 1.5:
        predictions6.append(1)
    else:
        predictions6.append(2)


preds8 = []
```

```
for index, row in X_test.iterrows():
    preds8.append(predict(row, rules_depth8))

predictions8 = []
for i in preds8:
    if i < 0.5:
        predictions8.append(0)
    elif i >= 0.5 and i < 1.5:
        predictions8.append(1)
    else:
        predictions8.append(2)


preds10 = []
for index, row in X_test.iterrows():
    preds10.append(predict(row, rules_depth10))

predictions10 = []
for i in preds10:
    if i < 0.5:
        predictions10.append(0)
    elif i >= 0.5 and i < 1.5:
        predictions10.append(1)
    else:
        predictions10.append(2)
```

```
preds12 = []
for index, row in X_test.iterrows():
    preds12.append(predict(row, rules_depth12))

predictions12 = []
for i in preds12:
    if i < 0.5:
        predictions12.append(0)
    elif i >= 0.5 and i < 1.5:
        predictions12.append(1)
    else:
        predictions12.append(2)
```

The predictions of the test dataset with each decision tree created are compared to the true outcomes of the test dataset. The accuracy of each decision tree on the test dataset is found by taking the sum of the observations where the predicted class is equal to the actual class and dividing it by the total number of observations in the test dataset. Finally, this is multiplied by 100 to produce a percentage out of 100%. The accuracy of each decision tree is found below.

```
print("The accuracy of the tree with maximum depth 2 is: %f%%"
      % (((sum((y_test == predictions2)**2)/len(y_test))*100)))
```

```
The accuracy of the tree with maximum depth 2 is: 93.133137%
```

```
print("The accuracy of the tree with maximum depth 4 is: %f%%"
 % (((sum((y_test == predictions4)**2)/len(y_test))*100)))
```

```
The accuracy of the tree with maximum depth 4 is: 96.076078%
```

```
print("The accuracy of the tree with maximum depth 6 is: %f%%"
      % (((sum((y_test == predictions6)**2)/len(y_test))*100)))
```

```
The accuracy of the tree with maximum depth 6 is: 96.493070%
```

```
print("The accuracy of the tree with maximum depth 8 is: %f%%"
      % (((sum((y_test == predictions8)**2)/len(y_test))*100)))
```

```
The accuracy of the tree with maximum depth 8 is: 97.042059%
```

```
print("The accuracy of the tree with maximum depth 10 is: %f%%"
      % (((sum((y_test == predictions10)**2)/len(y_test))*100)))
```

```
The accuracy of the tree with maximum depth 10 is: 97.372053%
```

```
print("The accuracy of the tree with maximum depth 12 is: %f%%"
      % (((sum((y_test == predictions12)**2)/len(y_test))*100)))
```

```
The accuracy of the tree with maximum depth 12 is: 97.576048%
```

Based on the accuracy of each decision tree presented above, it follows that for this dataset, the optimal depth of the decision tree is found to be a maximum depth of 12. Since this is the tree with the largest maximum depth, it is possible that the true optimal decision tree (the decision tree with the true highest accuracy) has a larger maximum depth than 12. However, we will consider this to be the optimal tree when taking into account other external variables, such as the time it takes to run the code on this dataset. Due to the size of the dataset and its complexity, the time it takes to create the decision tree increases with a large maximum depth. Therefore, in this case, the optimal decision tree is the tree with maximum depth 12, which has an accuracy of 97.576048%.

✓ 0s completed at 11:05 PM