莊晴雯 Michelle Novenda Hartono
108062281

1. **Implement the Decision Tree**

- Load the Example Data

  In this part, I did a little bit of modification for the values in the WaitEstimate feature. Since the range of the numbers is too big, I preprocessed the data by dividing the values with 35 (floor division to avoid decimal values). This way, I can classify the data better and represent the values 3, 6, 7, 8, 12, 20 as 0; 40 as 1; 70 and 80 as 2.

- Calculating the Entropy

  *def partition(data, column, threshold):*

  I first get the values of the corresponding column and convert it to a list for better processing, then create two empty lists called matchIdx and falseIdx to store the indexes of the data that matches the assigned column's threshold and the one that doesn't. I also have the variable curIdx to iterate through the elements and keep track of the current index. When iterating through the elements in the corresponding column, I have two conditions to check:

  - If the value of the current element is smaller or equal to the threshold.
    - ⇨ Append curIdx (current index) to matchIdx.
  - Else (if the value of the current element is greater than the threshold).
    - ⇨ Append curIdx (current index) to falseIdx.

  Then increment curIdx by 1 in each iteration.

  Afterwards, we can use iloc to select the data that match and the data that doesn't match using the matchIdx and falseIdx index lists and assign the value to match_branch and false_branch afterwards.

  *def entropy(data):*

  First, we collect the information of the length of the data, the values of the Wait feature, and the numbers of T and F in the Wait feature column. Next we calculate probs (the probability of T), which can be found by dividing the number of T in wait and the length of the data. However, if the length of the data is 0, the probability will automatically be set to 0.

  The formula to calculate entropy is:

  $-$ probT $\log_2 probT$ $-$ probF $\log_2 probF$

  We can use np.log2 to calculate the log and 1-probs to get the value of probF. We do have to take the case of when probs equals to 0 and probs equals to 1 into account. We do this to avoid errors in np.log2.

  - If probs == 0 => Directly set probT $\log_2 probT$ to 0.
  - If probs == 1 => Directly set probF $\log_2 probF$ to 0.

- Find the Best Split

  What I did here is to first find the column_best, then find the value_best of the column_best afterwards. Beforehand, I stored the name of the features in a list called colList. To find the column_best, I preprocessed the data and create a data structure in this format:

  [ array([ [#T, #F], [#T, #F] ]) … array([ [#T, #F], [#T, #F], … , [#T, #F] ]) ]

  Each array in the list represents the features, and the list inside each arrays represents the branches of the corresponding feature. The array can contain more than 2 lists, depending on the number of the branches it has. Each list contains the number of True and the number of False values according to the Wait feature. I then calculate the entropy of each features and store them in a list in order. After having the value of entropy for each features, I can do comparison to get the value of the smallest entropy and store it in the variable all_entropy. Since the entropies are stored in order in the list, I can use the index function to get the index of the smallest entropy, which is also the index of the best feature in the feature column list.

  After knowing the column_best, we can now get the best threshold of the column_best. To do this, we first sort the dataframe based on the column_best value. Next we need to find the mid value for each distinct value that appears consecutively in the column_best column. I stored each mid value that can be found into a list called thresList. Afterwards, I calculated the Information Gain for each element in thresList using the formula:

  entropy(data) – (entropy(leftBranch) + entropy(rightBranch))

  where we get leftBranch and rightBranch by doing partition(sortedData, column_best, thres).

  sortedData: the sorted dataframe; thres: each element in the thresList

  I then store each value of the Information Gain into a list called infoGainList. If the list is empty, then value_best will be 0, but if it is not empty, then we must find the largest value of the Information Gain, get the index in infoGainList, then use the index to get value_best from thresList.

- <u>Decision Tree Building</u>

  To implement this with recursion, we have the base case:

  ⇨ return decisionSubTree, features, thresholds if depth == 0, column_best == 0 or value_best == 0, enLeft == 0 or enRight == 0.

  (enLeft: entropy of left subtree, enRight: entropy of right subtree).

  We can find the values of column_best and value_best by calling findBestSplt(df).

  The recursive calls would be:

  ⇨ Calling buildTree for left decision subtree and right decision subtree and decreasing the depth by 1 in each recursive calls.

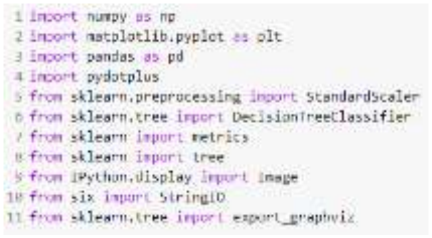
  The left decision subtree and right decision subtree can be found by calling the function partition(df, column_best, value_best).

  Then we need to compare the left decision subtree's Information Gain with each element of the leftThresholds list. If the value of the Information Gain is greater than or equal to the current element, and the current element, length of the leftFeatures list, and the length of the leftThresholds list is not equal to 0, then we can extend the thresholds list with leftThresholds and features list with leftFeatures. Lastly, we need to break to avoid extending unnecessary elements to the list.

  We do the same thing for rightThresholds and rightFeatures. If right decision subtree's Information Gain is greater than or equal to the current element of the rightThresholds list, and if the current element is not equal to 0, as well as the length of the rightFeatures list and the rightThresholds list, then we can extend the thresholds list with rightThresholds and the features list with rightFearures, and break from the loop.

## 2. Classification with the MIMIC Dataset

To implement this part, I imported a few modules:



```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pydotplus
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn import tree
from IPython.display import Image
from six import StringIO
from sklearn.tree import export_graphviz
```

Top 3 splitting features and their thresholds: mvar12 (0.5), CMO (0.5), mvar23 (0.5)

Preprocess the MIMIC data in the second part: Removed the subject_id and indextime column while processing the data and doing the prediction since these two datasets did not affect the prediction.

Decision Tree construction and code explanation:

- <u>Prediction (Performance)</u>

  I first saved the subject_id column for x_train and x_test into a new dataframe which will be used later. Then I preprocessed the data by dropping the columns subject_id and indextime from x_train and x_test since these data does not have any impact on the hospDIED data. Next, I used DecisionTreeClassifier and set the max_depth to 4. I can then use dt.fit(x_train, y_train) to build the decision tree classifier from the x and y training set. Afterwards, I can call dt.predict(x_test) to predict the value of y_test, which is stored in y_pred. I then added back the subject_id column to x_train and x_test using pd.concat so that I can pair the prediction to the corresponding subject_id and export the prediction as a CSV without any errors.

- <u>Visualizing the Decision Tree</u>

  Again, I dropped the subject_id column since it does not affect the hospDIED prediction, and called DecisionTreeClassifier, set the value of max_depth to 4, and call dt.fit(x_train, y_train). I then put the feature names to a list, and use StringIO() to initialize the data stream which will be used for the out_file attribute in export_graphviz. Next, I used the export_graphfiz method to export the decision tree in DOT format and it will then be written into out_file. I passed dt as the decision tree to be exported to GraphViz, dotData as the out_file, features list as the feature_names, and class_names as [‘1’, ‘0’] since there are only these two values in hospDIED prediction result. I then used pydotplus.graph_from_dot_data(dotData.getvalue()) to visualize the decision tree from the dot data, and assign it to graph. Lastly, I just need to call write_png to create the png file of the visualized decision tree.