MTL782 DATA MINING

Assignment 1

Team Members

Khushvind Maurya 2021MT10238
Aniket Singh 2021MT10256
Rishabh Jaiswal 2021MT10924

Contents

Link to Colab Notebook:
https://colab.research.google.com/drive/1fgndrUZsIUPLQUVdp4dExW4I-H4X57BT?usp=sharing

1 Task 1 & 2

1.1 Loading the the provided excel file and converting it into a Pandas dataframe

```
[ ] df = pd.read_excel('Data-RF.xlsx')
    df.head()
```

	Lab-Test1(30)	Lab-Test2(24)	Midsem Test (90)	Gender	Attendance	Grade
0	13.00	24	66.0	Male	High	A
1	15.00	24	67.0	Female	High	A
2	5.25	24	45.0	Male	High	B-
3	2.75	19	34.0	Male	High	C-
4	7.25	24	30.0	Male	High	C-

1.2 Encoding the categorical data

TF-DF requires classification labels to be integers in [0, num_labels), so we convert the label column 'Grade' from strings to integers.

```
[7] classes_grd = df["Grade"].unique().tolist()
    print(f"Grade classes: {classes_grd}")
    df["Grade"] = df["Grade"].map(classes_grd.index)
```

```
Grade classes: ['A', 'B-', 'C-', 'D', 'B', 'A-', 'C', 'E']
```

```
[8] encoded_df = df.copy()

    classes_gen = encoded_df["Gender"].unique().tolist()
    print(f"Gender classes: {classes_gen}")
    encoded_df["Gender"] = encoded_df["Gender"].map(classes_gen.index)

    classes_att = encoded_df["Attendance"].unique().tolist()
    print(f"Attendance classes: {classes_att}")
    encoded_df["Attendance"] = encoded_df["Attendance"].map(classes_att.index)
```

```
Gender classes: ['Male', 'Female']
Attendance classes: ['High', 'Low', 'Moderate']
```

1.3 Splitting the datasets and Converting to Tensorflow datasets

Note: To keep the results consistent everytime the notebook is run, we considered the splitting where first 70% data points go to train_df, and remaining 30% go to test_df. We also made a function to perform random splitting incase it is required.

```
[40] # Functions To Split Data
     def random_split_dataset(dataset, test_ratio=0.30):
       """Randomly splits a panda dataframe in two."""
       indices = df.index.tolist()
       test_size = int(len(df) * test_ratio)

       random.seed(42)
       test_indices = random.sample(population=indices, k=test_size)
       train_indices = [index for index in indices if index not in test_indices]

       train_df = df.loc[train_indices]
       test_df = df.loc[test_indices]
       return train_df, test_df

     def split_dataset(dataset, test_ratio=0.30):
       """Splits a panda dataframe in two (Not Random)."""
       n_samples = len(dataset)
       split_index = int(n_samples - test_ratio * n_samples)
       train_df = dataset.iloc[:split_index]
       test_df = dataset.iloc[split_index:]
       return train_df, test_df

     # Splitting the original df
     train_df, test_df = split_dataset(df)
     # Splitting the encoded df
     enc_train_df, enc_test_df = split_dataset(encoded_df)
```

```
[41] # Converting to Tensorflow datasets
     train_ds = tfdf.keras.pd_dataframe_to_tf_dataset(train_df, label="Grade")
     test_ds = tfdf.keras.pd_dataframe_to_tf_dataset(test_df, label="Grade")

     # Converting to Tensorflow datasets
     enc_train_ds = tfdf.keras.pd_dataframe_to_tf_dataset(enc_train_df, label="Grade")
     enc_test_ds = tfdf.keras.pd_dataframe_to_tf_dataset(enc_test_df, label="Grade")
```

1.4 Taining The Decision Tree Model to compare the accuracies of original dataset, and encoded dataset (num_trees = 1, for gradient boosted trees is a decision tree)

```
[42] model = tfdf.keras.GradientBoostedTreesModel(num_trees = 1, verbose = 0)
     model.fit(train_ds)
     enc_model = tfdf.keras.GradientBoostedTreesModel(num_trees = 1, verbose = 0)
     enc_model.fit(enc_train_ds)

     <keras.src.callbacks.History at 0x78df439a44f0>
```

```
[43] model.compile(metrics=["accuracy"])
     accuracy = model.evaluate(test_ds, return_dict=True, verbose = 0)["accuracy"]
     enc_model.compile(metrics=["accuracy"])
     enc_accuracy = enc_model.evaluate(enc_test_ds, return_dict=True, verbose = 0)["accuracy"]
     print ("Accuracy for a decision tree trained on orignal dataset is, ", accuracy)
     print ("Accuracy for a decision tree trained on encoded dataset is, ", enc_accuracy)

     Accuracy for a decision tree trained on orignal dataset is,  0.7933333516120911
     Accuracy for a decision tree trained on encoded dataset is,  0.7933333516120911
```

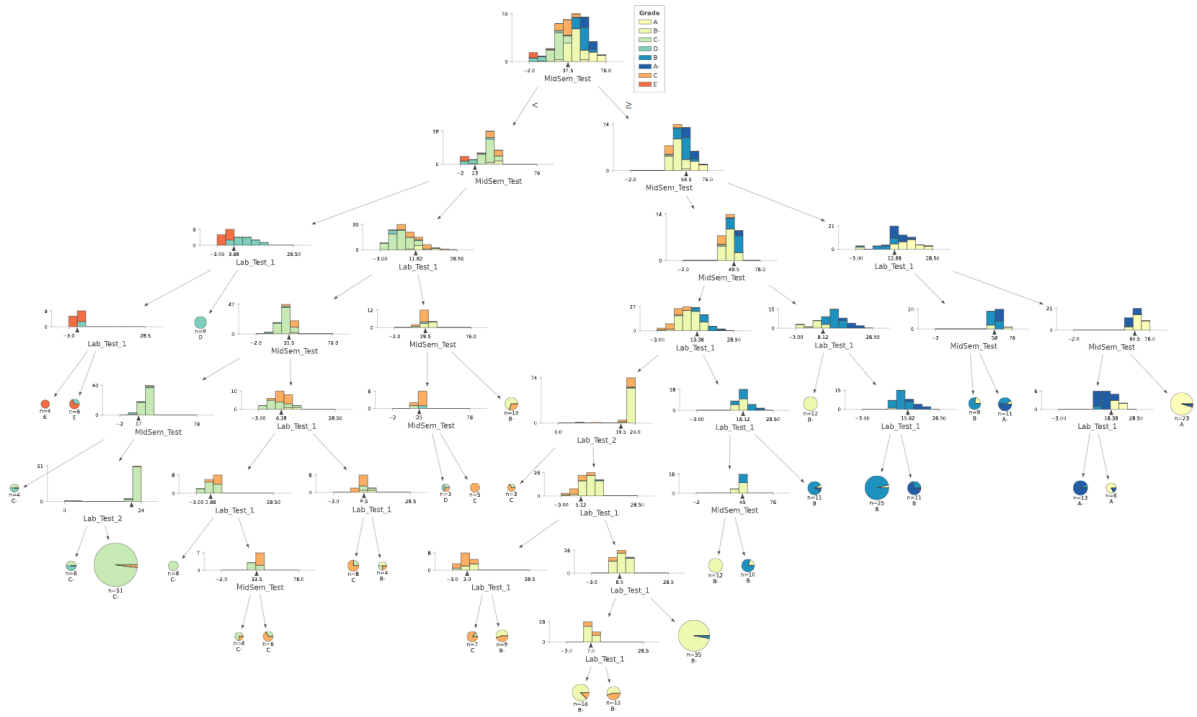
1.5 Training Random Forest Model with 30 DTs

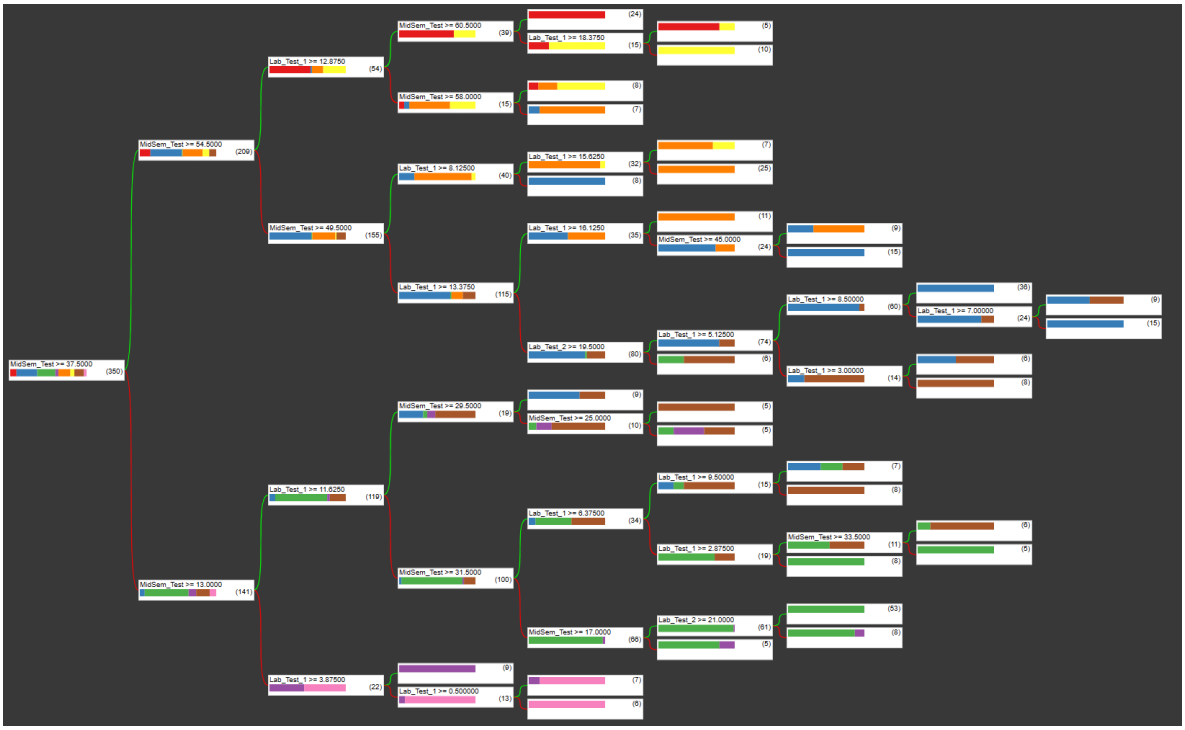
```
[44] model_rf = tfdf.keras.RandomForestModel(num_trees = 30, verbose = 0)
     model_rf.fit(train_ds)

     <keras.src.callbacks.History at 0x78df439eb310>
```

```
[45] model_rf.compile(metrics=["accuracy"])
     evaluation = model_rf.evaluate(test_ds, return_dict=True, verbose = 0)
     accuracy = evaluation["accuracy"]

     print ("Accuracy after training random forest model with 30 DTs = ", accuracy)

     Accuracy after training random forest model with 30 DTs =  0.8533333539962769
```

2 Task 3

2.1 Visualizing the first tree in the trained Random Forest Model





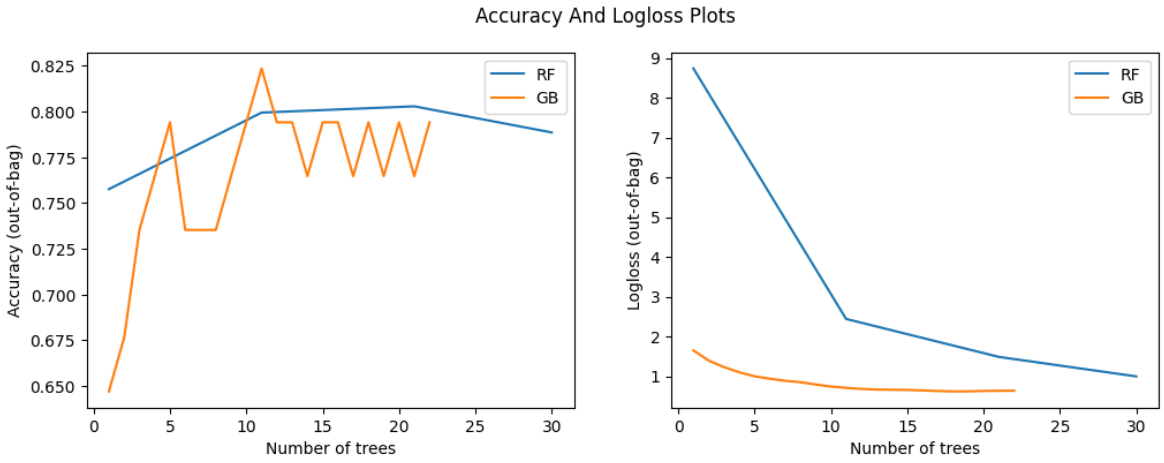
3 Task 4

3.1 Training Gradient Boosted Decision Trees with 30 DTs

```
[70] model_gb = tfdf.keras.GradientBoostedTreesModel(num_trees = 30, verbose = 0)
     model_gb.fit(train_ds)

     <keras.src.callbacks.History at 0x78de8cda71f0>

[71] model_gb.compile(metrics=["accuracy"])
     accuracy = model_gb.evaluate(test_ds, return_dict=True, verbose = 0)["accuracy"]

     print ("Accuracy after training gradient boosted model with 30 DTs, ", accuracy)

     Accuracy after training gradient boosted model with 30 DTs,  0.8799999952316284
```

3.2 Comparing Accuracies Of Random Forest and Gradient Boosted Trees

```
[63] model_rf = tfdf.keras.RandomForestModel(num_trees = 30, verbose = 0)
     model_gb = tfdf.keras.GradientBoostedTreesModel(num_trees = 30, verbose = 0)

     # Train the model.
     model_rf.fit(train_ds)
     model_gb.fit(train_ds)

     <keras.src.callbacks.History at 0x78de98fda410>

[64] model_rf.compile(metrics=["accuracy"])
     accuracy_rf = model_rf.evaluate(test_ds, return_dict=True, verbose = 0)["accuracy"]
     model_gb.compile(metrics=["accuracy"])
     accuracy_gb = model_gb.evaluate(test_ds, return_dict=True, verbose = 0)["accuracy"]

     print ("Accuracy of Random Forest Model is ", accuracy_rf)
     print ("Accuracy of Gradient Boosted Tree Model is ", accuracy_gb)

     Accuracy of Random Forest Model is  0.8533333539962769
     Accuracy of Gradient Boosted Tree Model is  0.8799999952316284
```

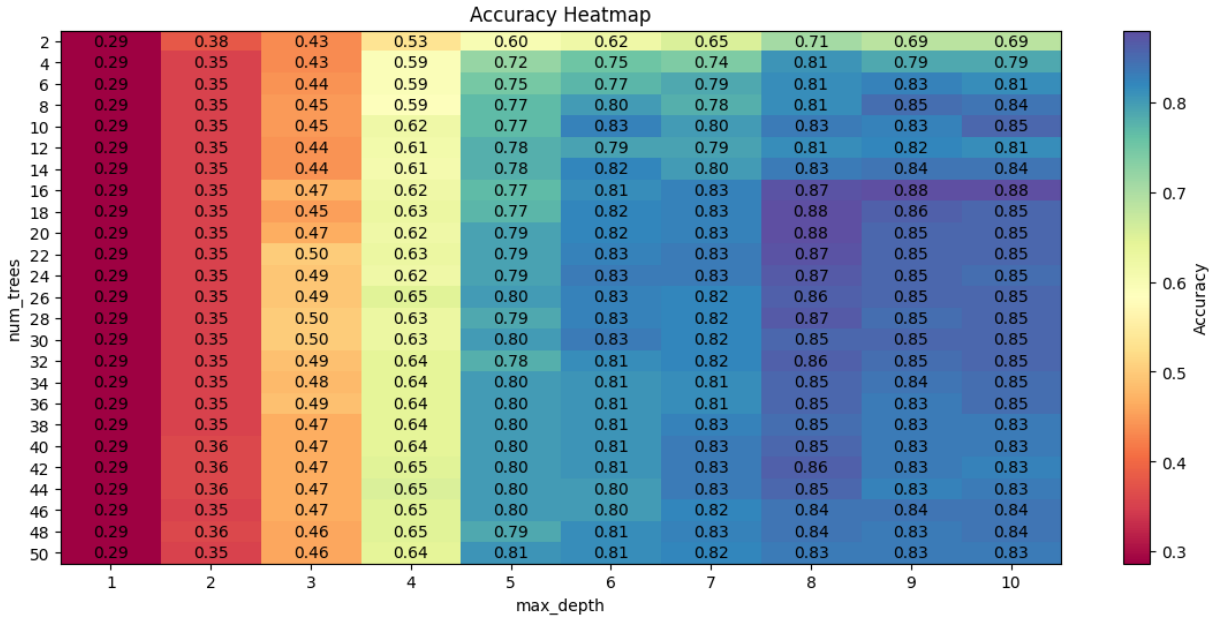


4 Task 5

4.1 Comparing the training and testing accuracies of Random Forest Model

```
[72] model = tfdf.keras.RandomForestModel(num_trees = 30, verbose = 0)

     # Train the model.
     model.fit(train_ds)

     <keras.src.callbacks.History at 0x78de8cd44f10>

[73] model.compile(metrics=["accuracy"])
     accuracy_train = model.evaluate(train_ds, return_dict=True, verbose = 0)["accuracy"]
     model.compile(metrics=["accuracy"])
     accuracy_test = model.evaluate(test_ds, return_dict=True, verbose = 0)["accuracy"]

     print ("Training Accuracy is ", accuracy_train)
     print ("Testing Accuracy is ", accuracy_test)

     Training Accuracy is  0.9285714030265808
     Testing Accuracy is  0.8533333539962769
```

4.2 Finding out the number of trees and maximum depth hyper-parameters for a reasonable accuracy of 85%. And finding out the impact of n-trees and max depth hyper-parameter on the performance of model.



From the above plot, we get accuracy of more than or equal to 85% for (num_trees, max_depth) value pairs of (16,8),(16,9),(18,8),(20,8) etc. There are obviously more (num_trees, max_depth) pairs with greater than 85% accuracy that are not shown in the plot.

We can also observe from the heatmap plot of accurcy that on increasing the 'num_trees' (number of trees) hyperparameter (for a given 'max_depth') the accuracy increases, until

it reaches a maximum, after which it decreases slightly, and then remains almost constant. A similar trend is observed with the 'max_depth' hyperparameter (for a given 'num_trees'). On increasing the 'max_depth' hyperparameter, the accuracy increases to a maximum, after which it slightly decreases. It will remain the same after a value of 'max_depth' because there is an upper limit to the depth of trees (As there are a finite number of attributes).