CSCE 633: Machine Learning Homework-2

Arunachalam Venkatachalam

7th March 2024

1 Information Gain

	X1=0	**X1=1**
Y=1	0	2
Y=2	1	1
Y=3	1	1

	X2=0	**X2=1**
Y=1	0	2
Y=2	1	1
Y=3	2	0

- Conditional entropy for X1:

$$\begin{aligned} H(Y|X1) = &- P(X1=0)\sum_{y=1}^{3} P(Y=y|X1=0)\log_2 P(Y=y|X1=0) \\ &- P(X1=1)\sum_{y=1}^{3} P(Y=y|X1=1)\log_2 P(Y=y|X1=1) \end{aligned}$$

$$P(X1=0) = \frac{2}{6}$$

$$P(X1=1) = \frac{4}{6}$$

$$P(Y=1|X1=0) = \frac{0}{2}$$

$$P(Y=2|X1=0)=\frac{1}{2}$$

$$P(Y=3|X1=0)=\frac{1}{2}$$

$$P(Y=1|X1=1)=\frac{2}{4}$$

$$P(Y=2|X1=1)=\frac{1}{4}$$

$$P(Y=3|X1=1)=\frac{1}{4}$$

$$\begin{aligned}H(Y|X1)=&-\frac{2}{6}\sum_{y=1}^{3}[\frac{0}{2}*\log_2(0)+\frac{1}{2}*\log_2(\frac{1}{2})+\frac{1}{2}*\log_2(\frac{1}{2})]\\&-\frac{4}{6}\sum_{y=1}^{3}[\frac{2}{4}*\log_2(\frac{2}{4})+\frac{1}{4}*\log_2(\frac{1}{4})+\frac{1}{4}*\log_2(\frac{1}{4})]\end{aligned}$$

$$\begin{aligned}H(Y|X1)&=\frac{2}{6}+1\\H(Y|X1)&=\frac{4}{3}\\H(Y|X1)&=1.33\end{aligned}$$

- Conditional entropy for X2:

$$\begin{aligned}H(Y|X2)=&-P(X2=0)\sum_{y=1}^{3}P(Y=y|X2=0)\log_2 P(Y=y|X2=0)\\&-P(X2=1)\sum_{y=1}^{3}P(Y=y|X2=1)\log_2 P(Y=y|X2=1)\end{aligned}$$

$$P(X2=0)=\frac{3}{6}$$

$$P(X2 = 1) = \frac{3}{6}$$

$$P(Y = 1|X2 = 0) = \frac{0}{3}$$

$$P(Y = 2|X2 = 0) = \frac{1}{3}$$

$$P(Y = 3|X2 = 0) = \frac{2}{3}$$

$$P(Y = 1|X2 = 1) = \frac{2}{3}$$

$$P(Y = 2|X2 = 1) = \frac{1}{3}$$

$$P(Y = 3|X2 = 1) = \frac{0}{3}$$

$$\begin{aligned} H(Y|X2) = &-\frac{3}{6}\sum_{y=1}^{3}[\frac{0}{3} * \log_2(0) + \frac{1}{3} * \log_2(\frac{1}{3}) + \frac{2}{3} * \log_2(\frac{2}{3})] \\ &-\frac{3}{6}\sum_{y=1}^{3}[\frac{2}{3} * \log_2(\frac{2}{3}) + \frac{1}{3} * \log_2(\frac{1}{3}) + \frac{0}{3} * \log_2(0)] \end{aligned}$$

$$H(Y|X2) = 0.4565 + 0.4565$$
$$H(Y|X2) = 0.913$$

- The overall entropy before splitting is:

$$H(Y) = -\frac{2}{6} * log_2(\frac{2}{6}) - \frac{2}{6} * log_2(\frac{2}{6}) - \frac{2}{6} * log_2(\frac{2}{6})$$

$$H(Y) = -\frac{2}{6} * (-1.585) - \frac{2}{6} * (-1.585) - \frac{2}{6} * (-1.585)$$

$$H(Y) = 1.585$$

- Information Gain of X1:

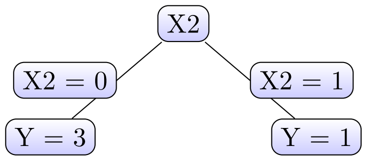
$$Info - gain(X1) = H(Y) - H(Y|X1) = 1.585 - 1.33 = 0.2516$$

- Information Gain of X2:

$$Info - gain(X2) = H(Y) - H(Y|X2) = 1.585 - 0.913 = 0.672$$

- Since the information gain of X2 is greater than X1, split on X2 is desired.

  Decision Tree:
  As we split the data based on X2, we see that when X2=0 Y=3 is the majority although we have a single point of Y=2. Similarly, we see that when X2=1 Y=1 is the majority although we have a single point of Y=2. So we will say that when X2=0 all datapoints belong to Y=3 class and when X2=1 all datapoints belong to Y=1 class. This discards further split based on X1. If we choose to not go by majority then we have to split X2=0 and X2=1 based on X1 which will not give a accurate good tree to split the data.



- An example with $X_1 = 0$ and $X_2 = 1$ will be classified as $Y = 1$ on this tree since $X_2 = 1$.

2 Classification Tree

Dataset Description
This dataset contains information about loan applicants, including their gender, marital status, number of dependents, education level, employment status, income, loan amount, loan term, credit history, property area, and loan status.

Each row represents a single applicant.

Data Imputation
The dataset appears to have some missing values which were imputed based on mean for continuous varaibles and based on most frequent entries for categorical variables. This may induce some bias but removing all the rows with missing values may not be the optimal way because the dataset is small.

Data Encoding
The dataset contains categorical variables with binary values, i.e. "Yes" or "No", "Graduated" or "Not Graduated", and categorical variables with distinct classes (2 or 3) in case of Property Area - "Rural", "Urban", "Semi Urban".
Whereas, to train a classfication tree numerical inputs are required. Thus encoding of the categorical data is mandatory. A mapping between Categorical variable and a set of numeric values were established and the data was transformed to a numeric based dataset.

Missing Data
Yes, the data has some missing values.

Column	**Total Blank Rows**
Loan_ID	0
Gender	12
Married	3
Dependents	13
Education	0
Self_Employed	29
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	17
Loan_Amount_Term	12
Credit_History	43
Property_Area	0
Loan_Status	0

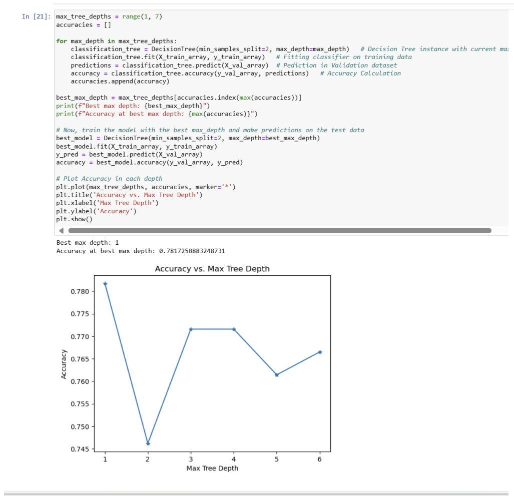
Total Blank places: 129

Data preparation
Removing Loan_ID feature as it won't generalize the data well as it is unique, and could add complexity of the model. Using Gender, Married, Dependents, Education, Self_Employed, ApplicantIncome, CoapplicantIncome, LoanAmount, Loan_Amount_Term, Credit_History, Property_Area as the features based on which the tree will split.

Different feature types in this data

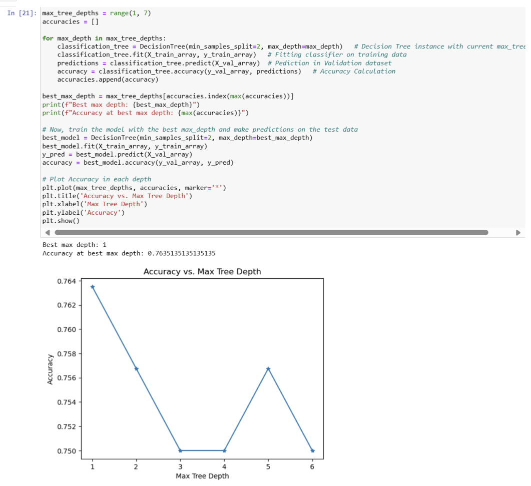
Variable	Type	Description
Loan_ID	Categorical	Unique identifier for the loan
Gender	Categorical	Gender of the applicant
Married	Categorical	Marital status of the applicant
Dependents	Categorical	Number of dependents of the applicant
Education	Categorical	Education level of the applicant
Self_Employed	Categorical	Self-employment status of the applicant
Property_Area	Categorical	Area where the property is located
Loan_Status	Categorical	Status of the loan (Approved/Rejected)
ApplicantIncome	Continuous	Income of the applicant
CoapplicantIncome	Continuous	Income of the coapplicant
LoanAmount	Continuous	Amount of the loan
Loan_Amount_Term	Continuous	Term of the loan
Credit_History	Continuous	Credit history of the applicant

Trouble with Max depth, I always get a higher accuracy at a max depth of 1 for different splits of the data. This is shown in the below screenshots. (had a word with TA, TA asked me to put up the screenshots showing that I get a better accuracy for max depth 1 in all cases)



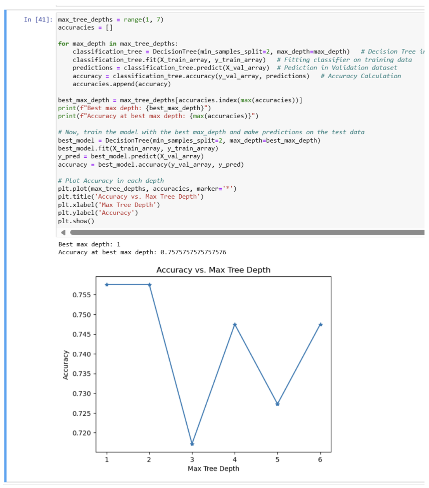
```
In [21]: max_tree_depths = range(1, 7)
         accuracies = []

         for max_depth in max_tree_depths:
             classification_tree = DecisionTree(min_samples_split=2, max_depth=max_depth)   # Decision Tree instance with current ma
             classification_tree.fit(X_train_array, y_train_array)   # Fitting classifier on training data
             predictions = classification_tree.predict(X_val_array)  # Pediction in Validation dataset
             accuracy = classification_tree.accuracy(y_val_array, predictions)   # Accuracy Calculation
             accuracies.append(accuracy)

         best_max_depth = max_tree_depths[accuracies.index(max(accuracies))]
         print(f"Best max depth: {best_max_depth}")
         print(f"Accuracy at best max depth: {max(accuracies)}")

         # Now, train the model with the best max_depth and make predictions on the test data
         best_model = DecisionTree(min_samples_split=2, max_depth=best_max_depth)
         best_model.fit(X_train_array, y_train_array)
         y_pred = best_model.predict(X_val_array)
         accuracy = best_model.accuracy(y_val_array, y_pred)

         # Plot Accuracy in each depth
         plt.plot(max_tree_depths, accuracies, marker='*')
         plt.title('Accuracy vs. Max Tree Depth')
         plt.xlabel('Max Tree Depth')
         plt.ylabel('Accuracy')
         plt.show()

         Best max depth: 1
         Accuracy at best max depth: 0.7817258883248731
```

(a) Max Depth 1 (train-60 val- 40)



```
In [21]: max_tree_depths = range(1, 7)
         accuracies = []

         for max_depth in max_tree_depths:
             classification_tree = DecisionTree(min_samples_split=2, max_depth=max_depth)   # Decision Tree instance with current max_tree
             classification_tree.fit(X_train_array, y_train_array)   # Fitting classifier on training data
             predictions = classification_tree.predict(X_val_array)  # Pediction in Validation dataset
             accuracy = classification_tree.accuracy(y_val_array, predictions)   # Accuracy Calculation
             accuracies.append(accuracy)

         best_max_depth = max_tree_depths[accuracies.index(max(accuracies))]
         print(f"Best max depth: {best_max_depth}")
         print(f"Accuracy at best max depth: {max(accuracies)}")

         # Now, train the model with the best max_depth and make predictions on the test data
         best_model = DecisionTree(min_samples_split=2, max_depth=best_max_depth)
         best_model.fit(X_train_array, y_train_array)
         y_pred = best_model.predict(X_val_array)
         accuracy = best_model.accuracy(y_val_array, y_pred)

         # Plot Accuracy in each depth
         plt.plot(max_tree_depths, accuracies, marker='*')
         plt.title('Accuracy vs. Max Tree Depth')
         plt.xlabel('Max Tree Depth')
         plt.ylabel('Accuracy')
         plt.show()

         Best max depth: 1
         Accuracy at best max depth: 0.7635135135135135
```

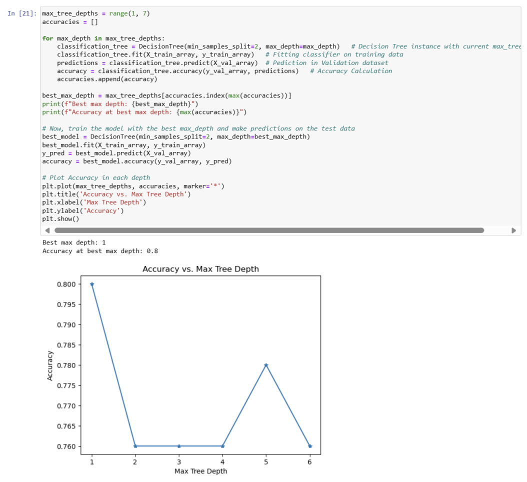
(b) Max Depth 1 (train-70 val- 30)



```
In [41]: max_tree_depths = range(1, 7)
accuracies = []

for max_depth in max_tree_depths:
    classification_tree = DecisionTree(min_samples_split=2, max_depth=max_depth)   # Decision Tree i
    classification_tree.fit(X_train_array, y_train_array)   # Fitting classifier on training data
    predictions = classification_tree.predict(X_val_array)  # Pediction in Validation dataset
    accuracy = classification_tree.accuracy(y_val_array, predictions)   # Accuracy Calculation
    accuracies.append(accuracy)

best_max_depth = max_tree_depths[accuracies.index(max(accuracies))]
print(f"Best max depth: {best_max_depth}")
print(f"Accuracy at best max depth: {max(accuracies)}")

# Now, train the model with the best max_depth and make predictions on the test data
best_model = DecisionTree(min_samples_split=2, max_depth=best_max_depth)
best_model.fit(X_train_array, y_train_array)
y_pred = best_model.predict(X_val_array)
accuracy = best_model.accuracy(y_val_array, y_pred)

# Plot Accuracy in each depth
plt.plot(max_tree_depths, accuracies, marker='*')
plt.title('Accuracy vs. Max Tree Depth')
plt.xlabel('Max Tree Depth')
plt.ylabel('Accuracy')
plt.show()
```

```
Best max depth: 1
Accuracy at best max depth: 0.7575757575757576
```

(a) Max Depth 1 (train-80 val- 20)



```
In [21]: max_tree_depths = range(1, 7)
accuracies = []

for max_depth in max_tree_depths:
    classification_tree = DecisionTree(min_samples_split=2, max_depth=max_depth)   # Decision Tree instance with current max_tree
    classification_tree.fit(X_train_array, y_train_array)   # Fitting classifier on training data
    predictions = classification_tree.predict(X_val_array)  # Pediction in Validation dataset
    accuracy = classification_tree.accuracy(y_val_array, predictions)   # Accuracy Calculation
    accuracies.append(accuracy)

best_max_depth = max_tree_depths[accuracies.index(max(accuracies))]
print(f"Best max depth: {best_max_depth}")
print(f"Accuracy at best max depth: {max(accuracies)}")

# Now, train the model with the best max_depth and make predictions on the test data
best_model = DecisionTree(min_samples_split=2, max_depth=best_max_depth)
best_model.fit(X_train_array, y_train_array)
y_pred = best_model.predict(X_val_array)
accuracy = best_model.accuracy(y_val_array, y_pred)

# Plot Accuracy in each depth
plt.plot(max_tree_depths, accuracies, marker='*')
plt.title('Accuracy vs. Max Tree Depth')
plt.xlabel('Max Tree Depth')
plt.ylabel('Accuracy')
plt.show()
```

```
Best max depth: 1
Accuracy at best max depth: 0.8
```

(b) Max Depth 1 (train-90 val- 10)

Figure 2: Depth vs Accuracy plots for different data split

3 Boosting

Finding the Optimal Lambda

The process of finding the optimal lambda value for L2 regularization involves evaluating the model's performance (e.g., AUC) with different lambda values.

- Define Lambda Values: Define a list of lambda values to explore.
- Cross-Validation: For each lambda value, perform K-fold cross-validation with bootstrapping. This helps mitigate the effect of randomness in data splits.
- Evaluate Model Performance: Within each fold, train an XGBoost model with the current lambda and evaluate its performance on the validation fold using AUC.
- Calculate Mean AUC: For each lambda, compute the average AUC across all folds. This provides a more robust estimate of the model's performance.
- Select Optimal Lambda: Finally, identify the lambda value that leads to the highest mean AUC. This is considered the optimal lambda for L2 regularization as it balances model complexity and generalization.

Classification Trees vs Boosting

Accuracy of Classification Tree = 75.75%
Accuracy of XGBoost = 68.68%

The classification tree performs better than boosting. The reason for the better performance of classification trees than XGBoost:

- Could be due to innate nature of decision tree to overfit the data.
- XGBoost performs better on larger datasets, since the dataset is small decision trees could have performed better.
- Dataset may have simple relationships that a decision tree can capture effectively without the need for the complexity that XGBoost offers.
- Suboptimal performance of XGBoost due to ineffective tuning of hyper-parameters.