Report

Objective: To design a Decision Tree classifier to predict which employees will leave next.

Tree Node Structure
In the implementation of decision tree, we have created a Node class.
The decision tree is an n-ary tree.
It has following associated attributes:

`root`	defines the root of tree
`pos`	number of positives at that node
`neg`	number of negatives at that node
`children`	a dictionary of the root's children. the key is subtree node's name and value is subtree object.
`result`	a boolean 0 or 1 depending on whether number of positives is greater or negatives
`tf`	a boolean true false which tells whether the node is leaf or not

If an employee is leaving the company, i.e. `left` col is 1, it is considered to be positive, otherwise negative.

Preparing Data
Randomly sampled the dataset.
Divided the dataset into 2 parts: 80% training, 20% validation
The target column is `left`.

1. Part-1

Problem : To train decision tree only on categorical data and find performance measures.

Solution

1. Select the categorical attributes:
   - Work_accident
   - promotion_last_5years
   - sales
   - salary

2. Find the impurity score of training label distribution.
   The impurity measured used is Entropy.
   Entropy is calculated as follows:
   `Entropy = E(Si) = -(q log(q) + (1-q) log(1-q))`
   where, q is the probability of an employee leaving the company.
   `entropyCalculate(dataframe, col_name)` function performs this step.
   Arguments - `dataframe`: the training dataset, `col_name`: target column, left
   Returns - total entropy

3. Now compute entropy for each unique value of candidate attributes.

Suppose the candidate attribute is salary, then we compute entropy for each of its unique values , i.e. low, medium and high.
The total entropy is then sum of weights of unique attributes multiplied with their entropy

$$I(S,A)=\sum_i (|\frac{Si}{S}|\circ E(Si))$$

where A is an attribute.
`entropyAttribute(dataframe, col_name, label)` function calculates entropy for each unique sub-attribute.
Arguments- `dataframe`: training dataset, `col_name`: attribute column, `label`: target column
Returns- entropy attribute

4. Compute Information Gain (reduction in impurity score) provided by candidate attribute.

$$Gain(S,A)=E(S)-I(S,A)$$

Select attribute which provides maximum gain. It is made the root node.
`findMaxInfoGain(dataframe)` returns the maximum root and its entropy.
Arguments- `dataframe`: training dataset.
Returns- root node and its entropy

5. Now we can build the tree.
`buildTree(dataframe)` function is the main function which returns the tree model.
It selects the root node, and makes it a Node object.
For each of the root ‘s unique value, it extracts its corresponding rows in a new temporary dataframe.
e.g. if the root node is salary, then for each unique attribute, low, medium and high, we select the corresponding rows.
Then drop this column and recursively call the `buildTree` function again.
Base Condition:
   - If we reach a pure node, i.e., either number of positives or negatives become 0, a leaf node is returned with corresponding decision.
   - If only one column is remaining, i.e the target `‘left’` column, then also we return a leaf node with decision depending on number of positives and negatives.

6. To start validating, we call the `helper_validate(df,root, predict_col)` function.
Arguments- `df`: validate dataset, `root`: root of the tree, `predict_col`: name of column in which predictions will be stored
We select each row of the dataframe and traverse the tree accordingly. When a leaf node is reached it returns the result stored there.

7. Then we calculate performance measures using `findMeasures` function.
   - True Positive: if target is 1 and predicted is also 1
   - True Negative: if target is 0 and predicted is also 0.
   - False Positive: if target is 0, but predicted is 1
   - False Negative: if target is 1, but predicted is 0.

$$Accuracy=\frac{(TP+TN)}{(TP+TN+FN+FP)}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 - Score = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

8. Prediction on sample test file.
   Now we use the `predict(model,model_args,X)` function.
   Arguments: `model`: root of tree, `model args`: name of target column for sample test file, `X`: sample test csv file
   Here we make the predictions for sample file and store the results in a target column.

Observations:
- **Accuracy**: 77.1352313167%
- **Prediction**: 100.0%
- **Recall**: 0.194174757282%
- **F1-Score**: 0.00387596899225

---

2. Part-2

Problem: To train decision tree on categorical and numerical data and find performance measures.

Solution
1. Select the entire dataframe.

2. Calculate entire training dataframe entropy using `entropyCalculate(dataframe, col_name)` function.

3. If the column is a categorical one, compute entropy for each unique value of candidate attributes using `entropyAttribute(dataframe, col_name, label)` function.
   In case of a numerical attribute, split the column at an appropriate point.
   To select that point, we do the following:
   - For each unique value `x` of that column, divide the dataframe into 2 sets, one contains all rows less than than equal to `x` and another contains all rows greater than `x`.
   - Then we find weighted entropy for both the sets.
   - Choose the minimum entropy and store its unique value x. That value `x` is the `split` point for that column.

   The function `findSplit(dataframe, col)` does this task.

4. Use `findMaxInfoGain(dataframe,num, categ)` to make the root. Additionally it also returns the `split` point for numerical attributes. `Num` and `categ` are lists of numerical and categorical attributes.

5. Then we use `buildTree(dataframe, num, categ)` to build the tree.

It selects the root node, and makes it a `Node` object.
If the root node is numerical, we split it into two halves at the `split` point and then recursively call the `buildTree function.`
In case for categorical, it is same as the previous one.

6. To start validating, we call the `helper_validate(df,root, predict_col)` function. It is similar to previous part's `predict`.

7. Prediction on sample test file is same as Part1

Observations:
- **Accuracy:** 97.5088967972%
- **Prediction**: 92.8846153846%
- **Recall**: 96.2151394422%
- **F1-Score**: 0.945205479452

Results:
It can be inferred that after spending more than 6 years, employees do not leave the company. This can be used by the company to analyse which senior employees will be leaving next.

time_spend_company	2	3	4	5	6	7	8	9	10
Number of employees leaving	11	218	128	146	34	0	0	-	0

---

3. Part – 3

Problem: Contrast the effectiveness of Misclassification rate, Gini as impurity measures in terms of precision, recall and accuracy

Solution:
Here the impurity measures used are different.

$$Gini = 2 * q * (1 - q)$$

$$MisClassification = min(q, 1 - q)$$

Replacing the entropy formula with other impurity formulae, results can be calculated in a similar manner.

Observations:

Measure:	Entropy	Gini	MisClassification Rate
Accuracy	97.5088967972%	99.4217081851%	97.9982206406%

Precision	92.8846153846%	98.3486238532%	92.037037037%
Recall	96.2151394422%	99.2592592593%	99.2592592593%
F1 Score	0.945205479452	0.98801843318	0.956689124158

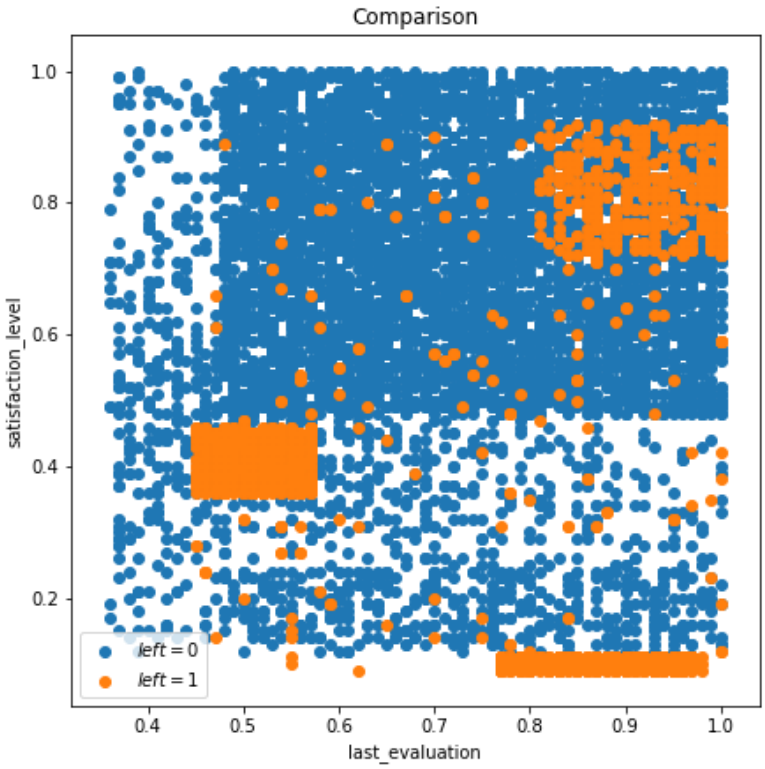
---

4. Part – 4

Problem: Visualize decision boundaries.

Solution:
After trying various combinations of features, `satisfaction_level` and `last_evaluation` were chosen as the two attributes for visualizing decision boundaries.
First, separate the `left = 0` and `left = 1` rows of both the features.
Then, plot scatter graphs with values where both the features had `left = 1` as one group, and both features as `left = 0` as second group.

Observation
Here 6823 employees have not left the company and 2167 have left.



It can be concluded that employees with high `last_evaluation` but low `satisfaction_level` tend to leave the company.

Other two groups for employees leaving the company are:

- Satisfaction level in the range of 0.3 to 0.45 and last evaluation 0.45 to 0.6.
- Satisfaction level in the range of 0.7 to 0.9 and last evaluation 0.8 to 1

It can be inferred that employees with `satisfaction_level` more than 0.9 do not leave the company.

---

5. Part – 5

Problem: Plot a graph of training and validation error with respect todepth of your decision tree. Also plot the training and validation error with respect to number of nodes in the decision tree

Solution

First build the tree and calculate the predictions.
Then, for each depth find the error at that depth. Store the errors in a list.
`Error = 1 - Accuracy`
Similarly, we count the number of nodes at each depth. Store the number of nodes in a list.
Then we plot a graph of errors with corresponding number of nodes , and errors with corresponding depth of the tree.
First compute errors for training dataset then for validation.

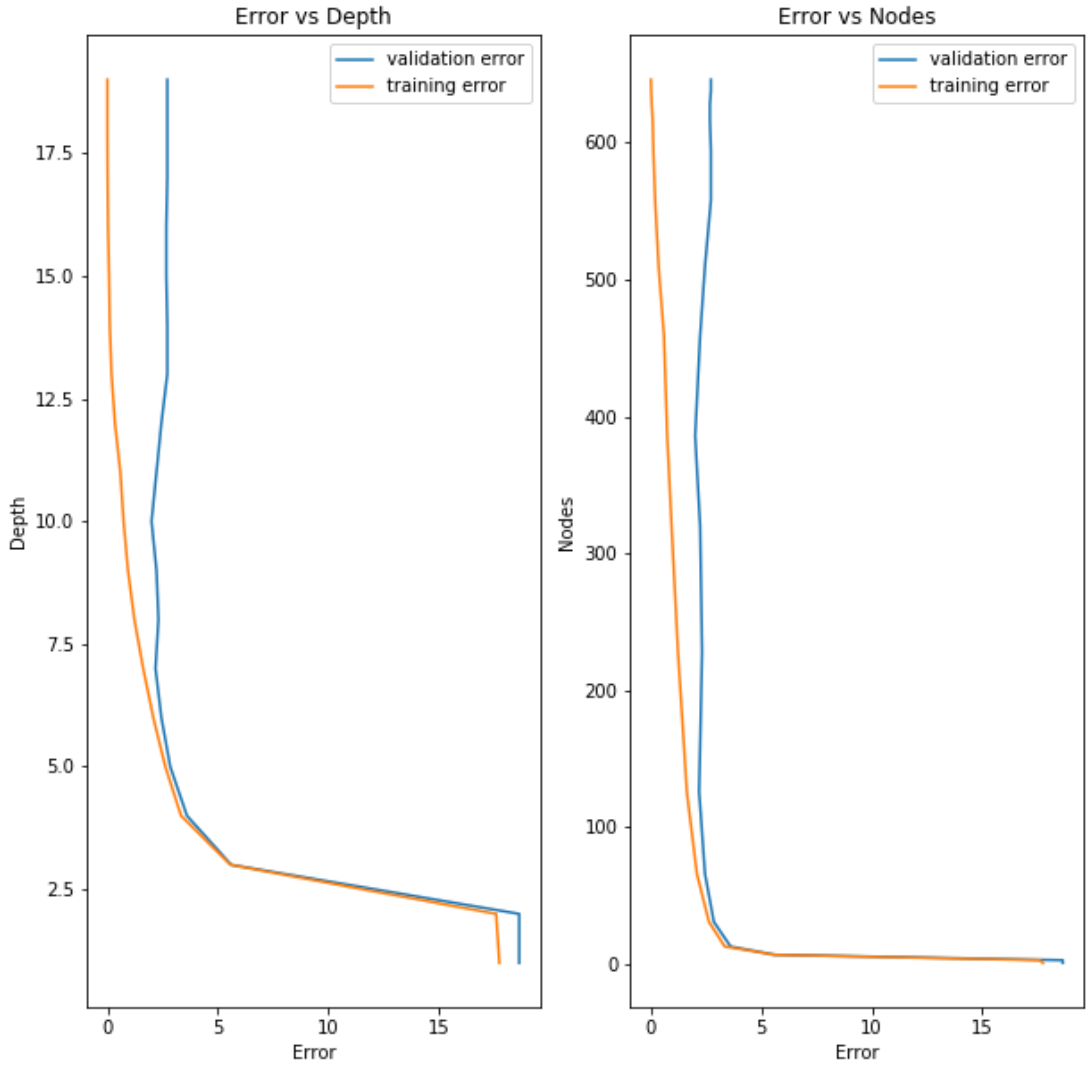
Observation

Depth: 18
Nodes: 629

Depth	Nodes	Error in Validation (%)	Error in Training (%)
1	1	18.327402135231324	17.81979977753059
2	3	18.861209964412808	17.541713014460512
3	7	5.293594306049826	5.650723025583986
4	13	3.247330960854086	3.4371523915461637
5	31	2.535587188612098	2.6362625139043416
6	76	2.40213523131672	2.113459399332598
7	138	2.135231316725978	1.6462736373748612
8	222	2.0017793594306	1.2791991101223488
9	301	1.912811387900362	0.9788654060066762

10	382	2.090747330960852	0.6896551724137936
11	455	2.0017793594306	0.46718576195773664
12	504	2.090747330960852	0.32258064516128115
13	544	1.957295373665474	0.2113459399332669
14	581	2.090747330960852	0.14460511679644128
15	601	2.0017793594306	0.07786429365962988
16	615	1.957295373665474	0.044493882091217074
17	623	1.957295373665474	0.0
18	629	1.957295373665474	0.0

The graph obtained is:



---

6. Part – 6

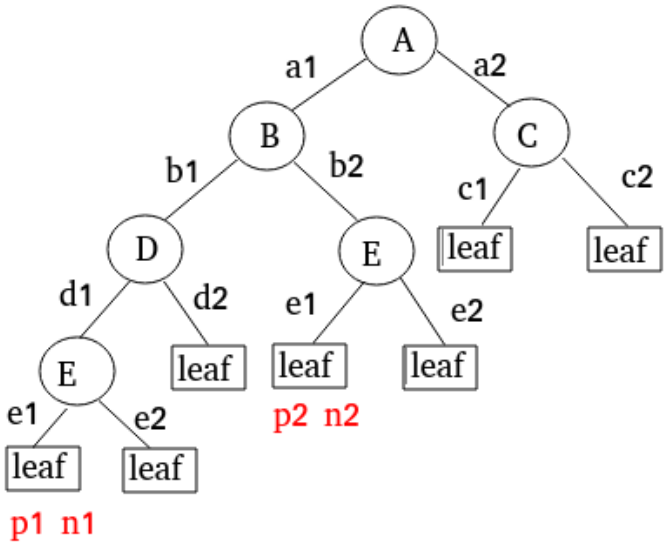
Problem : Explain how decision tree is suitable handle missing values(few attributes missing in test samples) in data

Solution

* One way is to store number of positives and negatives at each node.

Suppose there are 5 nodes [`A, B, C, D, E`] and the row is [a1, - , d1, e1]. Here attribute for column B is missing. So we compare the positives and negatives at node `A` and return the result accordingly.

That is in general, we compare the number of positives and negatives at the last non – missing node.



* Another way to handle the above case is as follows. Here also number of positives and negatives at each node is stored.
Since we don't know whether branch b1 or b2 is to be selected, so we traverse both branches of B. In the left subtree, we follow the path b1, d1, e1. For the right sub tree, we follow b2, e1. Then we compare the positives and negatives at both the leaves.
If p1 + p2 > n1 + n2, we decide it is positive, else negative.

* Another approach is to ignore the row where missing attributes are present.

* Also, we can fill in the missing values with some measure of mean, median or mode.