Exercise 2.3

1.1 Problem Statement:

Explore the problem of overfitting in decision tree and develop solution using pruning technique.

1.2 Description of Machine Learning Algorithm:

Decision Trees are an important type of algorithm for predictive modelling machine learning. Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

CART is a decision tree algorithm and it stands for Classification and Regression Trees. CART can be used for classification or regression predictive modelling problems. The representation for the CART model is a binary tree. It builds decision trees based on the attribute selection measure "Gini's Impurity Index". CART is an umbrella word that refers to the following types of decision trees:

Classification Trees: These are used to forecast the value of a categorical target variable.

Regression trees: These are used to forecast the value of a continuous target variable.

Advantages of Decision Trees:

- Decision trees are simple to understand, interpret, visualize.
- Decision trees implicitly perform variable screening or feature selection.
- Decision trees require little data preparation.
- Decision trees can handle both numerical and categorical data.
- Decision trees can also handle single-class and multi-class classification problems.
- Nonlinear relationships between parameters do not affect tree performance.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by Boolean logic.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

Disadvantages of Decision Trees:

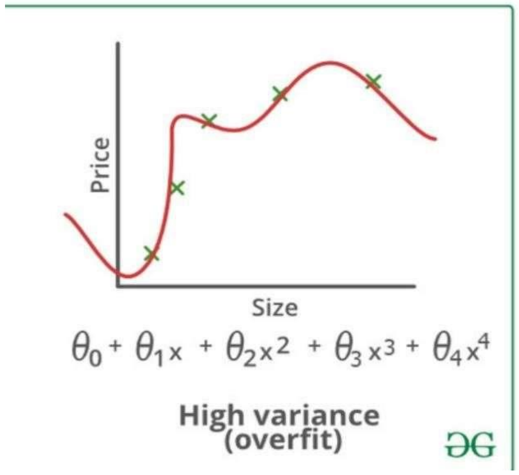
- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This is called variance, which needs to be lowered by methods like bagging, boosting etc.
- Predictions of decision trees are neither smooth nor continuous, but are piecewise constant approximations. Therefore, they are not good at extrapolation.

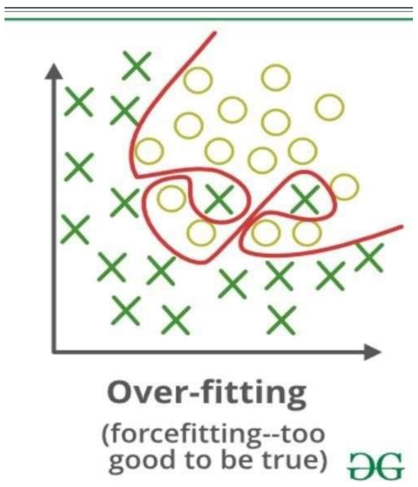
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the data set prior to fitting with the decision tree.

OVERFITTING Algorithm:

A statistical model is said to be overfitted when we train it with a lot of data (just like fitting ourselves in oversized pants!). When a model gets trained with so much data, it starts learning from the noise and inaccurate data entries in our data set. Then the model does not categorize the data correctly, because of too many details and noise. The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore, they can really build unrealistic models. A solution to avoid overfitting is using a linear algorithm if we have linear data or using the parameters like the maximal depth if we are using decision trees.

Examples:





Using Grid Search CV to prune the decision tree model to handle the problem of over fitting:

What is Grid Search CV?

GridSearchCV is a library function that is a member of sklearn’s model_selection package. It helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, you can select the best parameters from the listed hyperparameters.

In addition to that, you can specify the number of times for the cross-validation for each set of hyperparameters.

Defining hyper parameters:

1. estimator: estimator object you created
2. params_grid: the dictionary object that holds the hyperparameters you want to try
3. scoring: evaluation metric that you want to use, you can simply pass a valid string/ object of evaluation metric
4. cv: number of cross-validation you have to try for each selected set of hyperparameters
5. verbose: you can set it to 1 to get the detailed print out while you fit the data to

   GridSearchCV

6. n_jobs: number of processes you wish to run in parallel for this task if it -1 it will use all available processors.

sklearn.model_selection.GridSearchCV

Method : class sklearn.model_selection.GridSearchCV(estimator, param_grid, *, scoring
=None, n_jobs=None, refit=True, cv=None, verbose=0, pre_dispatch='2*n_jo
bs', return_train_score=False)
error_score=nan,

Important members are fit, predict.

GridSearchCV implements a “fit” and a “score” method. It also implements “score_samples”, “predict”, “predict_proba”, “decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used.

Parameters:

- estimator: estimator object

  This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a score function, or scoring must be passed.
- param_grid: dict or list of dictionaries

  Dictionary with parameters names (str) as keys and lists of parameter settings to try as values, or a list of such dictionaries, in which case the grids spanned by each dictionary in the list are explored. This enables searching over any sequence of parameter settings.
- Scoring: str, callable, list, tuple or dict, default=None

  Strategy to evaluate the performance of the cross-validated model on the test set.
- n_jobs:int, default=None

cvint, cross-validation generator or an iterable, default=None

Determines the cross-validation splitting strategy. Possible inputs for cv are:

- None, to use the default 5-fold cross validation,
- integer, to specify the number of folds in a (Stratified)KFold,
- CV splitter,
- An iterable yielding (train, test) splits as arrays of indices.

For integer/None inputs, if the estimator is a classifier and y is either binary or multiclass, StratifiedKFold is used. In all other cases, KFold is used. These splitters are instantiated with shuffle=False so the splits will be the same across calls.

Number of jobs to run in parallel. None means 1 unless in a joblib.parallel_backend context. -1 means using all processors. See Glossary for more details.

- return_train_score: bool, default=False

  If False, the cv_results_ attribute will not include training scores. Computing training scores is used to get insights on how different parameter settings impact the overfitting/underfitting trade-off. However computing the scores on the training set can be computationally expensive and is not strictly required to select the parameters that yield the best generalization performance.

1.3 Description of Data Set:

Title of the data set: Social Network Ads Dataset

The Social Network Ads data set contains information about the users who have whether purchased or not. The data contains 2 classes i.e., 1 or 0 where 1 represents the purchased class and 0 represents not purchased class. There are 400 data items in the data set. User id contains unique values.

Data Set characteristics: Multivariable

Number of Sample(or Instances) in the data set: 400

Number of Attributes or (Features): 5 Attribute Information:

1. User Id
2. Gender
3. Age
4. Estimated Salary
5. Purchased:

   ---0

   ---1
6. Number of Users Purchased : 143
7. Number of Users not purchased: 257
8. Predicted Attribute : Class of Purchased
9. Missing Attributes: NONE

1.4 Data Preprocessing and Exploratory Data Analysis:

Data preprocessing is the process of transforming raw data into an understandable format. It is also an important step in data mining as we cannot work with raw data. The quality of the data should be checked before applying machine learning or data mining algorithms.

Major Tasks in Data Preprocessing:

1. Data cleaning
2. Data integration
3. Data reduction
4. Data transformation

Exploratory data analysis (EDA) is a technique that data professionals can use to understand a dataset before they start to model it. Some people refer to EDA as data exploration. The goal of conducting EDA is to determine the characteristics of the dataset. Conducting EDA can help data analysts make predictions and assumptions about data. Often, EDA involves data visualization, including creating graphs like histograms, scatter plots and box plots.

Major Tasks in EDA:

1. Observe your dataset
2. Find any missing values
3. Categorize your values
4. Find the shape of your dataset
5. Identify relationships in your dataset
6. Locate any outliers in your dataset

1.5 Machine Learning Package Used for Model building:

The scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It provides simple and efficient tools for predictive data analysis. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

The sklearn.model_selection.train_test_split() method splits arrays or matrices into random train and test subsets. The parameters for the method are as follows:

```
sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, rand
om_state=None, shuffle=True, stratify=None)
```

It returns lists containing train-test split of inputs.

The sklearn.tree.DecisionTreeClassifier is a class capable of performing multi-class classification on a dataset. It takes as input two arrays: an array X, holding the training samples, and an array Y holding the class labels for the training samples. In case that there are multiple classes with the same and highest probability, the classifier will predict the class with the lowest index amongst those classes. As an alternative to outputting a specific class, the probability of each class can be predicted, which is the fraction of training samples of the class in a leaf. DecisionTreeClassifier is capable of both binary (where the labels are [-1, 1]) classification and multiclass (where the labels are [0, …, K-1]) classification. The parameters of the DecisionTreeClassifier class are as follows:

class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=N one, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_fea tures=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0)

The default values for the parameters controlling the size of the trees (e.g. max_depth, min_samples_leaf, etc.) lead to fully grown and unpruned trees which can potentially be very large on some data sets.

Limitations of sklearn.tree.DecisionTreeClassifier class:

- scikit-learn implementation does not support categorical variables for now.
- scikit-learn implementation can build only CART trees for now.

The DecisionTreeClassifier.fit() method builds a decision tree classifier from the training set (X, y). The parameters of the fit() method are as follows:

fit(X, y, sample_weight=None, check_input=True, X_idx_sorted='deprecated')

It returns an fitted Decision Tree estimator.
The DecisionTreeClassifier.predict() method predicts class or regression value for X. The parameters of the predict() method are as follows:

predict(X, check_input=True)

For a classification model, the predicted class for each sample in X is returned. For a regression model, the predicted value based on X is returned.

The sklearn.tree class supports visualization of decision trees. Once a Decision Tree Classifier is built it can be visualized in test representation using sklearn.tree.export_text() method. The parameters for the method are as follows:

sklearn.tree.export_text(decision_tree, *, feature_names=None, max_depth=10, spacing =3, decimals=2, show_weights=False)

Alternatively, the decision tree can be plotted sklearn.tree.plot_tree() method. The parameters for the method are as follows:

sklearn.tree.plot_tree(decision_tree, *, max_depth=None, feature_names=None, class_n ames=None, label='all', filled=False, impurity=True, node_ids=False, proportion=False, rounded=False, precision=3, ax=None, fontsize=None)

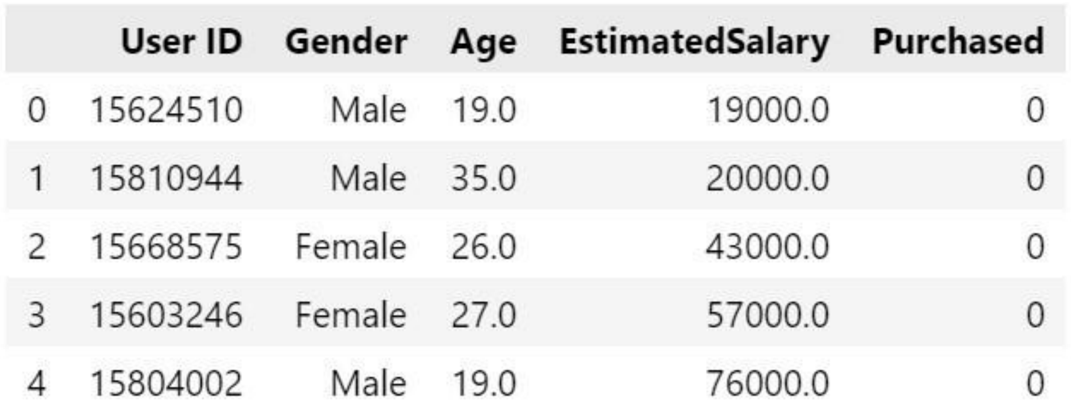
Once a Decision Tree Classifier is built, it can be evaluated for performance. The sklearn.metrics module implements several loss, score, and utility functions to measure classification performance. The sklearn.metrics.accuracy_score() method computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in y_true. The parameters for the method are as follows:

sklearn.metrics.accuracy_score(y_true, y_pred, *, normalize=True, sample_weight=None)

1.6 Implementation:

```
import numpy as np import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset dataset = pd.read_csv('Social_Network_Ads.csv')

# check no. of rows and columns in dataset

dataset.shape
O/P: (400, 5)

# display the first 5 rows of dataset

dataset.head()
```

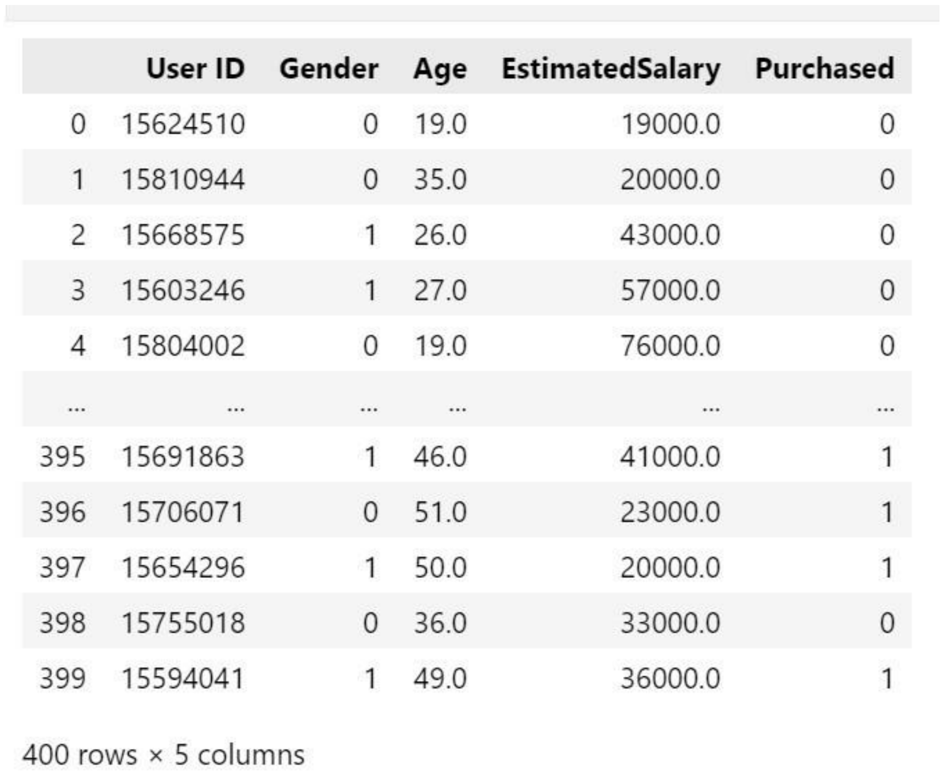
O/P:



	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19.0	19000.0	0
1	15810944	Male	35.0	20000.0	0
2	15668575	Female	26.0	43000.0	0
3	15603246	Female	27.0	57000.0	0
4	15804002	Male	19.0	76000.0	0

```
# sklearn DecisionTreeClassifier cannot handle categorical data.
# Hence, categorical attributes must be transformed into numerical attributes.

dataset['Gender'].replace(['Male', 'Female'],[0, 1], inplace = True)
dataset
```

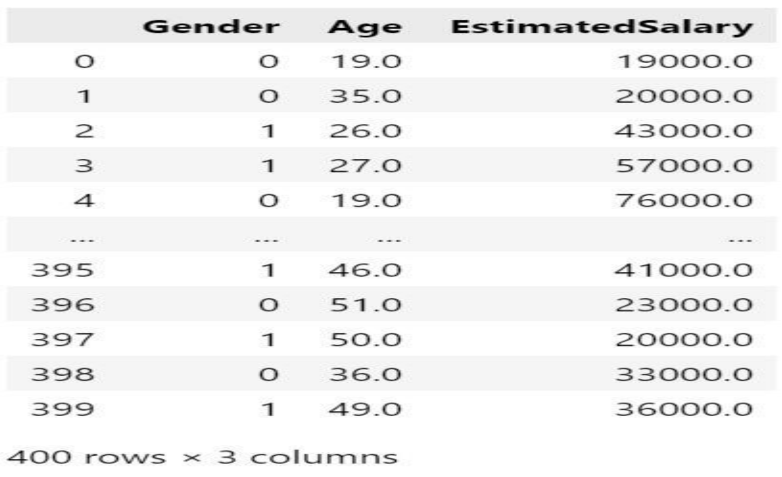
o/p:



	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	0	19.0	19000.0	0
1	15810944	0	35.0	20000.0	0
2	15668575	1	26.0	43000.0	0
3	15603246	1	27.0	57000.0	0
4	15804002	0	19.0	76000.0	0
...	...	...	...	...	...
395	15691863	1	46.0	41000.0	1
396	15706071	0	51.0	23000.0	1
397	15654296	1	50.0	20000.0	1
398	15755018	0	36.0	33000.0	0
399	15594041	1	49.0	36000.0	1

400 rows × 5 columns

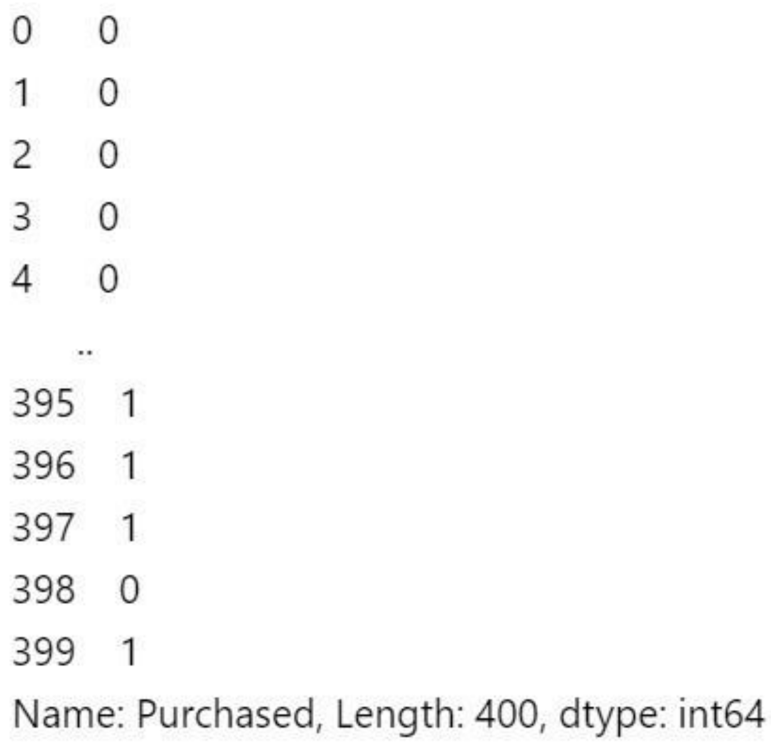
```
# check the unique values of 'Purchased' attribute
 dataset.Purchased.unique()
 o/p:
array([0, 1], dtype=int64)

# extracting the Independent variables into a dataframe
X = dataset.loc[:, ['Gender' , 'Age', 'EstimatedSalary']]
X
```



	Gender	Age	EstimatedSalary
0	0	19.0	19000.0
1	0	35.0	20000.0
2	1	26.0	43000.0
3	1	27.0	57000.0
4	0	19.0	76000.0
...	...	...	...
395	1	46.0	41000.0
396	0	51.0	23000.0
397	1	50.0	20000.0
398	0	36.0	33000.0
399	1	49.0	36000.0

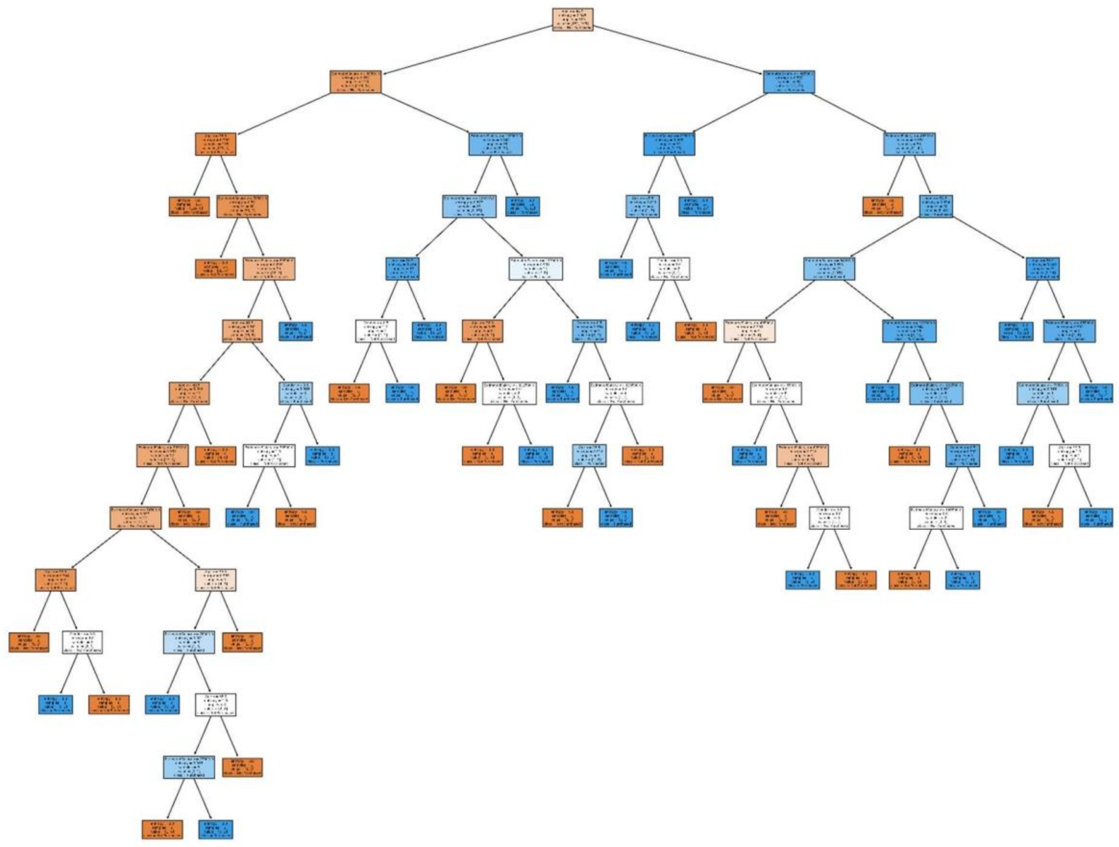
400 rows × 3 columns

```
# extracting the Target variables into a dataframe

y = dataset.loc[:, 'Purchased']   y  O/P:
```



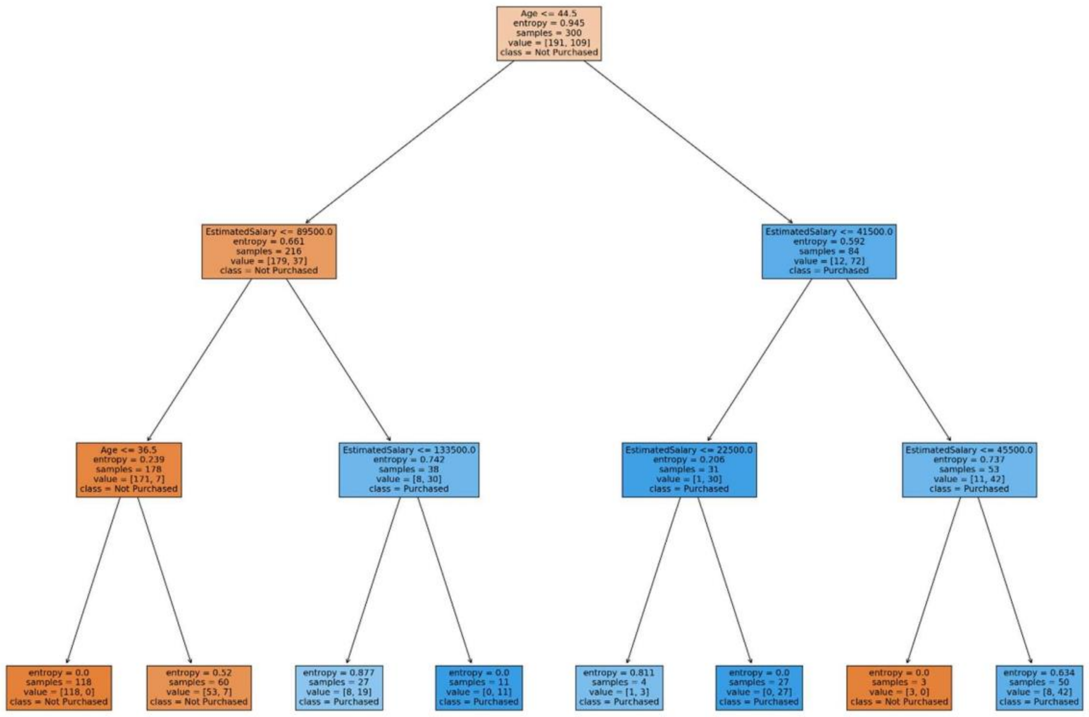
```
0      0
1      0
2      0
3      0
4      0
    ..
395    1
396    1
397    1
398    0
399    1
Name: Purchased, Length: 400, dtype: int64
```

```
  # Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X ,
y, test_size = 0.25,
random_state = 5)
# construct the decision tree model
from sklearn.tree import DecisionTreeClassifier

# create an instance "DecisionTreeClassifier" class
# set the criterion to be "entropy"
dtc = DecisionTreeClassifier(criterion = 'entropy' , random_state = 0)
# fit() method will construct the decision tree # by fitting
#the giventraining dataset.
dtc.fit(X_train , y_train)

# visualizing the decision tree.
from sklearn import tree
plt.figure(figsize = ( 25 , 20) , dpi = 300.0)
_ = tree.plot_tree(dtc, feature_names = ['Gender' , 'Age' ,
'EstimatedSalary'], class_names = ['Not Purchased', 'Purchased'] ,
filled = True )
```



```
# check the details of the tree

print ( 'Depth of the tree: ' , dtc.get_depth() ) print ( 'No. of leaves in the tree:'
dtc.get_n_leaves() )
O/P:
Depth of the tree: 13
No. of leaves in the tree: 47

# measure the accuracy of the constructed decision tree   from sklearn.metrics
import accuracy_score #
predict() method predicts classes for given dataset
# accuracy_score() will compute accuracy of model given
#                original class labels and predicted class labels.

# Let us first test the accuracy of the model on the training data itself.
 pred_train = dtc.predict(X_train)
 accuracy_train = accuracy_score(y_train, pred_train)   print('% of Accuracy on
training data: ', accuracy_train * 100)

# Let us test the accuracy of the model on the test data.
# Test data (or new data) is previously unseen by the model.
# Hence, its performance may reduce.
```

```
 pred_test = dtc.predict(X_test)   accuracy_test = accuracy_score(y_test,
pred_test)   print('% of Accuracy on test data: ', accuracy_test * 100)   O/P:
% of Accuracy on training data: 100.0
% of Accuracy on test data: 80.0


from sklearn.model_selection import GridSearchCV

# set the required parameters  params = {'criterion':
['entropy', 'gini'],
        'max_depth': range(1, 13)}

# create an instance of "GridSearchCV"  class

grid_search = GridSearchCV(estimator = dtc,
param_grid = params,scoring = 'accuracy', cv = 10, n_jobs = -1)

# Apply fit() method to construct different decision trees with given parameters  grid_search =
grid_search.fit(X_train, y_train)

# check the parameters of the best model

grid_search.best_estimator_   O/P:
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
# create an insance of "DecisionTreeClassifier" class with best parameters   best_dtc =
DecisionTreeClassifier(criterion = 'entropy' , max_depth=3,
random_state = 0)

# reconstruct the decision tree with best estimator parameters   best_dtc.fit(X_train ,
y_train)   O/P:
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
plt.figure(figsize = ( 25 , 20) , dpi = 200.0)
_ = tree.plot_tree( best_dtc, feature_names = ['Gender' , 'Age' , 'EstimatedSalary'],class_names = ['Not Purchased',
'Purchased'] , filled = True)
```



```
# check the details of the tree   print ( 'Depth of the tree after pruning: ' ,  best_dtc.get_depth() )
print ( 'No. of leaves in the tree after pruning: '  ,  best_dtc.get_n_leaves())   O/P:
Depth of the tree after pruning: 3
No. of leaves in the tree after pruning: 8
```

```
# test the accuracy of the model by using test dataset   pred_test =
best_dtc.predict(X_test)
 accuracy_test = accuracy_score(y_test, pred_test)
 print('% of Accuracy of the tree on test data after pruning : ', accuracy_test * 100)   O/P:
% of Accuracy of the tree on test data after pruning : 90.0
```

1.7 Results and Discussion:

- CART algorithm is implemented to construct decision tree.
- Upon analysing the decision tree it is noted that the decision tree has become too complex which has depth of 13 and the number of leaves in the tree is 47, this causes the problem of overfitting.
- It can be observed that the model performs almost perfect on the training dataset(100%). But its ability to make predictions on new data is less(80%). • using "GridSearchCV" for finding the optimal attribute measure and optimal depth of the decision tree.
- The maximum height of the tree is set to “3” with criterion as ‘entropy’.

- The decision is plotted after the pruning and it is observed that the No.of leaves in tree has decreased to “8”.
- Accuracy of the tree on test data after pruning is “90%”.

Hence ,To Explore the problem of overfitting in decision tree and develop solution using pruning technique is successful executed.