Machine Learning Report

Programming Assignment No. 1

Decision Tree & Bayesian (Naïve Bayes) Classifier

Name: Aryan Singh & Seemant Guruprasad Achari
Roll No :19CS30007 & 19CS30057

OBJECTIVES:

- Decision Tree
    - To split the dataset-C into 80%-20%.
    - Build a decision tree classifier using the ID3 algorithm.
    - Evaluating the classifier for 10 random splits.
    - Perform reduced error pruning ops. and analyze its efficacy.
    - Printing the tree after the prune operation.
    - Analyzing the performance variation with varying height.

- Bayesian (Naïve Bayes) Classifier
    - To split the dataset-c and encoding of the categorical features.
    - To find and drop samples having outliers feature value.
    - Training the Naive Bayes classifier and validating using a 10-fold validation method.
    - Applying the laplace correction on the same train and test split.
    - Evaluating the performance of the classifier.

Decision Tree

Task 1 : Data split and training

For splitting the dataset we implemented a python function with the following function prototype

def train_test_split(df,train_sample=0.5,target_col= LABEL)
Inputs : the dataframe, the training sample ratio (in our case .8), target_col (the class whose ratio would be maintained while splitting the data)

Maintaining the output class ratio ensures better training and testing.

The function returns train_df and test_df.

For training the classifier we used the following functions:-

def determineTypeOfFeature(df): Used for marking features as categorical or continuous.

def getPotentialSplits(featureName, trainData):
For continuous features the decisions were modeled as “less than equal” threshold decisions. The thresholds were found using this method. It returned a number of equispaced threshold values for a feature.

def getFeaturesList(trainData, featureSet):
This returns the list of categorical features and the list of continuous features with their threshold values.

For finding the best feature to split a node, we used the information gain method.
Using the following functions
def calculateTotalEntropy(trainData, label, classList):
def calculateEntropy(featureValueData, label, classList):
def calulateInformationGain(feature, trainData, label, classList):
def findMostImportantFeature(trainData, featureList, label, classList):

We calculate the entropy of the base class and based on the entropy of potential splitting we try to find out the best feature which will result in maximum information gain.

Class for decision tree and node

```
class DecisionNode:
   nodeId = 0 # this was used for numbering the nodes
   '''
   decisionThreshold and prePruningThreshold are design parameters.
   '''
   decisionThreshold = .35
   prePruningThreshold = 40
def fitData(self, trainData): # fits the data into the current nodes and its child if required
def predict(self, x_hat): # predicts the output based on the input x
def tempPrune(self): # temporarily prunes the node
def unPrune(self): # un prunes the temp prune nodes
def Prune(self):  # permanently prunes a node
def prePruning(self): # helps us decide whether to prune based on prePruningThreshold
def bestOption(self): # returns the best answer possible if we want to return from the current
node, it uses decision Threshold value to decide the answer.


class DecisionTree:
      max_depth = max_depth of the tree allowed
      root= root node of the decision tree

def fitData(self, trainData):
Fits the training data

def predict(self, x):
Predicts the class for a given input x

def getTreeDepth(self):
Returns the depth of the tree using dfs traversal.

def getTotalNodes(self):
Returns the total number of nodes in the tree
```

Evaluation metrics

def GetAccuracy(tree:DecisionTree, test_set, printDetails = True):
Returns the accuracy of the prediction

def GetPrecision(tree:DecisionTree, test_set, printDetails = True):
Returns the precision of predicting response = 1

def GetRecall(tree:DecisionTree, test_set, printDetails = True):
Returns the recall of predicting response = 1

Task - 1 output

Determining type of feature
Attr Gender is categorical
Attr Age is continuous
Attr Driving_License is categorical
Attr Region_Code is continuous
Attr Previously_Insured is categorical
Attr Vehicle_Age is categorical
Attr Vehicle_Damage is categorical
Attr Annual_Premium is continuous
Attr Policy_Sales_Channel is continuous
Attr Vintage is continuous

----------Task-1-Started----------
Accuracy = 0.8024072597486426 and Depth = 14
Precision = 0.3429070136464614 and Depth = 14
Recall   = 0.6702853598014888 and Depth = 14
The total number of nodes in the tree : 629
----------Task-1-Completed----------

Task 2

Trees were trained and tested over 10 random splits and the best tree along with its data set was stored for further tasks.

Task 2- Output

```
 ------------------------Task-2-Started--------------------------
Testing tree #0
Accuracy  = 0.8033564946652998 and Depth = 12
Precision = 0.33988533988533987 and Depth = 12
Recall    = 0.6436104218362283 and Depth = 12
# of nodes in tree 447

Testing tree #1
Accuracy  = 0.7896875118654365 and Depth = 12
Precision = 0.32910084157684927 and Depth = 12
Recall    = 0.6913771712158809 and Depth = 12
# of nodes in tree 473

Testing tree #2
Accuracy  = 0.7847894596954854 and Depth = 12
Precision = 0.3261735419630156 and Depth = 12
Recall    = 0.7112282878411911 and Depth = 12
# of nodes in tree 453

Testing tree #3
Accuracy  = 0.7934085127387326 and Depth = 12
Precision = 0.33293142426526 and Depth = 12
Recall    = 0.6851736972704715 and Depth = 12
# of nodes in tree 421

Testing tree #4
Accuracy  = 0.7924592778220754 and Depth = 12
Precision = 0.3306134783922635 and Depth = 12
Recall    = 0.6786600496277916 and Depth = 12
# of nodes in tree 452
```

Testing tree #5
Accuracy = 0.7964080950753692 and Depth = 12
Precision = 0.334057746041602 and Depth = 12
Recall   = 0.6674937965260546 and Depth = 12
# of nodes in tree 411

Testing tree #6
Accuracy = 0.7945096252420549 and Depth = 12
Precision = 0.32501599488163785 and Depth = 12
Recall   = 0.630272952853598 and Depth = 12
# of nodes in tree 460

Testing tree #7
Accuracy = 0.8028628925086381 and Depth = 12
Precision = 0.3405186385737439 and Depth = 12
Recall   = 0.6516749379652605 and Depth = 12
# of nodes in tree 459

Testing tree #8
Accuracy = 0.7921555226487451 and Depth = 12
Precision = 0.32792903028449066 and Depth = 12
Recall   = 0.6650124069478908 and Depth = 12
# of nodes in tree 432

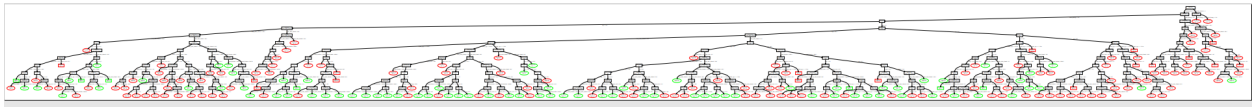
Testing tree #9
Accuracy = 0.7942438394653909 and Depth = 12
Precision = 0.3343896182284593 and Depth = 12
Recall   = 0.6873449131513648 and Depth = 12
# of nodes in tree 452

The Best tree : Root #0
Accuracy = 0.8033564946652998 and Depth = 12
Precision = 0.33988533988533987 and Depth = 12
Recall   = 0.6436104218362283 and Depth = 12
The total number of nodes in the tree : 447
----------Task-2-Completed----------

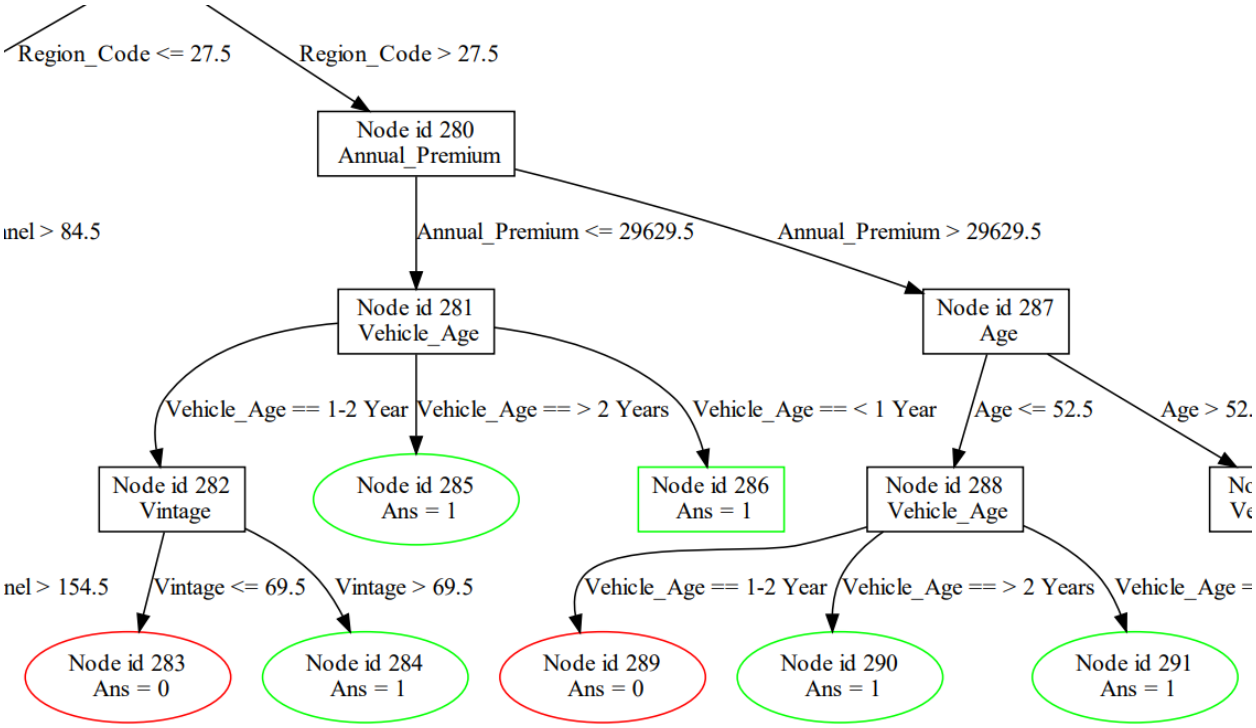
The best accuracy was found out to be 80.33% with a precision of 33% and recall of 64%. The total number of nodes was 447.

Tree diagrams

The tree looked like this:-



For better understanding here is a cropped snapshot of the tree diagram.



Here the Oval leaf node denotes that the node has been pruned either due to max depth constraint or due to pre pruning conditions. The rectangle leaf node denotes that it had samples of only one label class.

The green color signifies that the node is a true node. (it returns 1)
The red color signifies that the node is a false node. (it returns 0)

Task - 3

The difficulty that we faced while training and pruning was that the datasets had 87.5% 0 labeled responses. That is if a system naively returned 0 for every case, then its accuracy would be roughly 87.5%.

Only considering accuracy is not a good evaluation metric for the predictions.Our motivation was to develop a system which returns a list of customers who are likely to accept the offer presented by the sales team. Thus returning no customers is useless.

Without any additional constraints the pruning algorithm prunes root since, by pruning the node the majority of the samples are labeled 0 and the accuracy for that (87.5%) is better than 80.33%. Thus to prevent the system from pruning the root, we considered pruning only when recall was at least RECALL_LIMIT_THRESHOLD (= 0.25).

By adding the additional constraint we were able to improve its accuracy and precision.

The pruning steps were:-
For each node

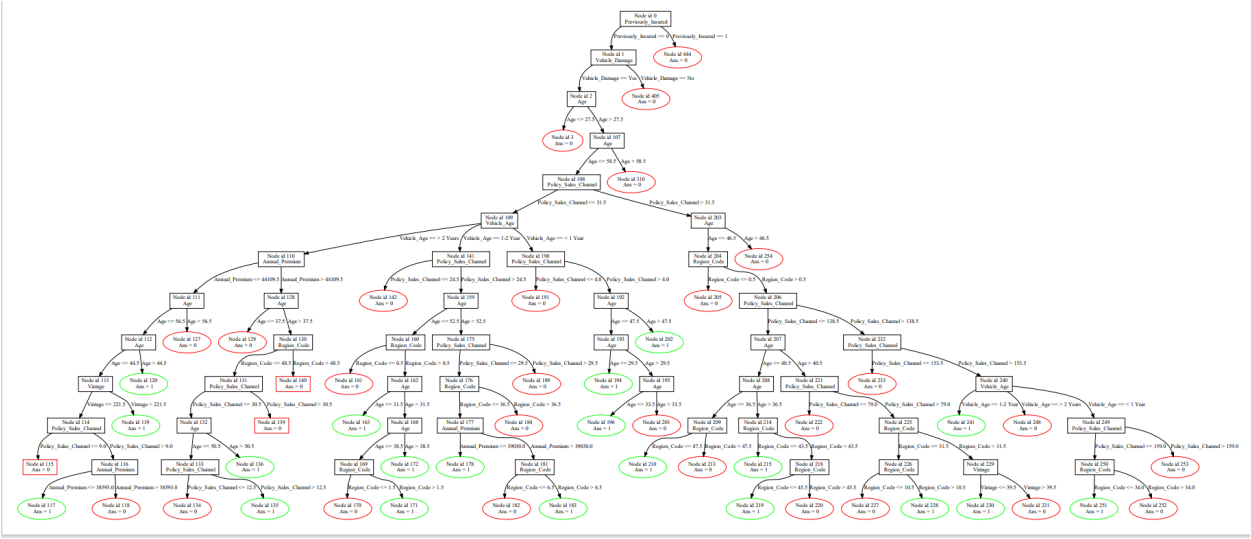
- Temporarily prune the node, if the recall is greater than threshold value, get its accuracy, update the best accuracy achieved till node.

If the best accuracy >= unpruned accuracy
Then permanently prune the node corresponding to the best accuracy.

By pruning we achieved better generalization and less complexity of the decision tree.

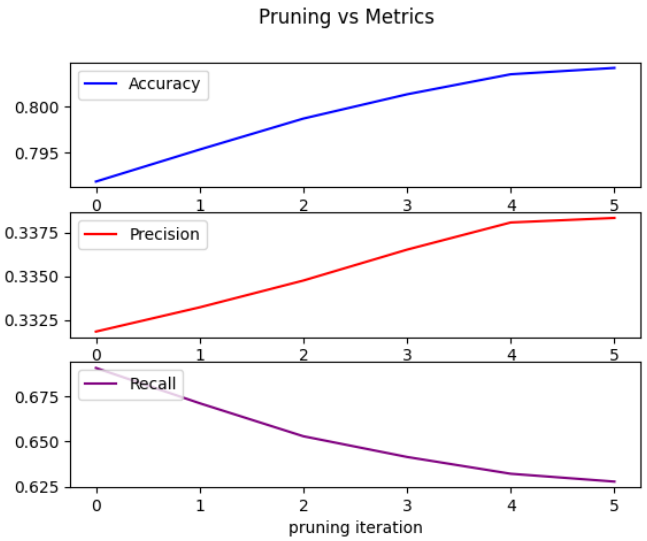
The total number of nodes in the tree : 97, which is significantly less than the original count of 447. The snapshot of the pruned tree :-

Diagrams and plots

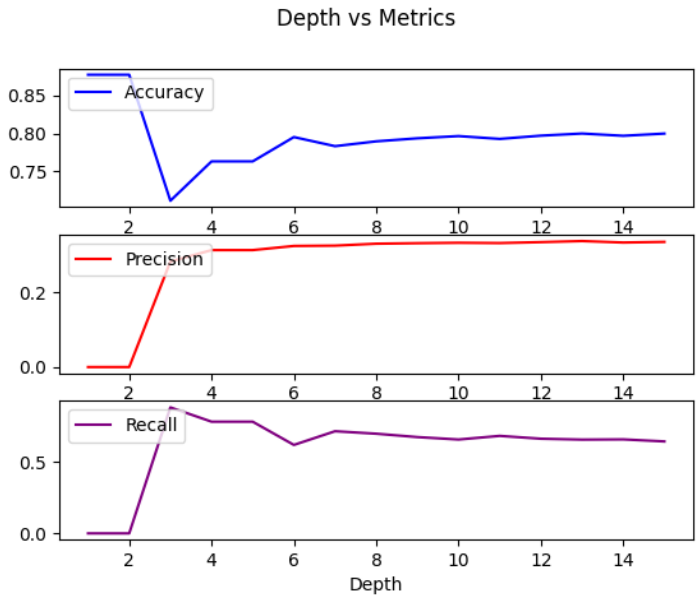


The plot for pruning iteration vs metrics : -

It is evident that the model accuracy improved along with its precision. And as precision improves the recall decreases since recall and precision are inversely related.



The plot for depth vs metrics :-



Firstly we should ignore the data collected for the depth <= 2, since this corresponds to a constant decision tree always returning 0.

For depth >2, we can see the general trend that as the complexity increases, the accuracy goes from .68 to .80, the precision also increases. And as precision increases the recall decreases.

For depth > 10, we can see that the model is not much improving and normally at this point models start overfitting as the complexity is increased further.

Task 3 outputs

```
-------------------------Task-3-Started---------------------------
Metrics before pruning
Accuracy  = 0.8033564946652998 and Depth = 12
Precision = 0.33988533988533987 and Depth = 12
Recall    = 0.6436104218362283 and Depth = 12
The total number of nodes in the tree : 447

Pruning started.
Base accuracy = 0.8100770778752325
Non decreasing pruning accuracy from 0.0 to 0.8100770778752325
Non decreasing pruning accuracy from 0.8100770778752325 to 0.8101150472718989
Non decreasing pruning accuracy from 0.8101150472718989 to 0.8120135171052132
Non decreasing pruning accuracy from 0.8120135171052132 to 0.8139119869385275
Non decreasing pruning accuracy from 0.8139119869385275 to 0.8139499563351938
Pruned nodeId = 3
Pruning iteration 1 completed
Base accuracy = 0.8139499563351938
Non decreasing pruning accuracy from 0.0 to 0.8139499563351938
Non decreasing pruning accuracy from 0.8139499563351938 to 0.8139879257318601
Non decreasing pruning accuracy from 0.8139879257318601 to 0.8158863955651745
Non decreasing pruning accuracy from 0.8158863955651745 to 0.8177848653984888
Pruned nodeId = 254
Pruning iteration 2 completed
Base accuracy = 0.8177848653984888
Non decreasing pruning accuracy from 0.0 to 0.8177848653984888
Non decreasing pruning accuracy from 0.8177848653984888 to 0.817822834795155
Non decreasing pruning accuracy from 0.817822834795155 to 0.8197213046284695
Non decreasing pruning accuracy from 0.8197213046284695 to 0.820594600751794
Pruned nodeId = 142
Pruning iteration 3 completed
Base accuracy = 0.820594600751794
Non decreasing pruning accuracy from 0.0 to 0.820594600751794
Non decreasing pruning accuracy from 0.820594600751794 to 0.8206325701484604
Non decreasing pruning accuracy from 0.8206325701484604 to 0.8225310399817747
Pruned nodeId = 310
Pruning iteration 4 completed
Base accuracy = 0.8225310399817747
Non decreasing pruning accuracy from 0.0 to 0.8225310399817747
Non decreasing pruning accuracy from 0.8225310399817747 to 0.822569009378441
Non decreasing pruning accuracy from 0.822569009378441 to 0.8229487033451038
Non decreasing pruning accuracy from 0.8229487033451038 to 0.823100580931769
Pruned nodeId = 184
Pruning iteration 5 completed
Base accuracy = 0.823100580931769
Non decreasing pruning accuracy from 0.0 to 0.823100580931769
Non decreasing pruning accuracy from 0.823100580931769 to 0.8231385503284353
Non decreasing pruning accuracy from 0.8231385503284353 to 0.8235182442950981
Pruned nodeId = 233
Pruning iteration 6 completed
Base accuracy = 0.8235182442950981
Non decreasing pruning accuracy from 0.0 to 0.8235182442950981
Non decreasing pruning accuracy from 0.8235182442950981 to 0.8235562136917645
Non decreasing pruning accuracy from 0.8235562136917645 to 0.8237080912784296
Pruned nodeId = 222
Pruning iteration 7 completed
```

Base accuracy = 0.8237080912784296
Non decreasing pruning accuracy from 0.0 to 0.8237080912784296
Non decreasing pruning accuracy from 0.8237080912784296 to 0.8237460606750959
Pruned nodeId = 405
Pruning iteration 8 completed
Base accuracy = 0.8237460606750959
Non decreasing pruning accuracy from 0.0 to 0.8237460606750959
Non decreasing pruning accuracy from 0.8237460606750959 to 0.8237840300717622
Pruned nodeId = 178
Pruning iteration 9 completed
Base accuracy = 0.8237840300717622
Non decreasing pruning accuracy from 0.0 to 0.8237840300717622
Pruned nodeId = 444
Pruning iteration 10 completed
Base accuracy = 0.8237840300717622
Non decreasing pruning accuracy from 0.0 to 0.7133310551695333
Non decreasing pruning accuracy from 0.7133310551695333 to 0.763830352735695
Non decreasing pruning accuracy from 0.763830352735695 to 0.7716520484489502
Non decreasing pruning accuracy from 0.7716520484489502 to 0.8019136575919809
Non decreasing pruning accuracy from 0.8019136575919809 to 0.8024831985419751
Non decreasing pruning accuracy from 0.8024831985419751 to 0.8109503739985572
Non decreasing pruning accuracy from 0.8109503739985572 to 0.8211261723051221
Non decreasing pruning accuracy from 0.8211261723051221 to 0.8221893154117781
Non decreasing pruning accuracy from 0.8221893154117781 to 0.8235941830884307
Non decreasing pruning accuracy from 0.8235941830884307 to 0.8237840300717622
Pruned nodeId = 241
Pruning iteration 11 completed
Base accuracy = 0.8237840300717622
Non decreasing pruning accuracy from 0.0 to 0.7133310551695333
Non decreasing pruning accuracy from 0.7133310551695333 to 0.763830352735695
Non decreasing pruning accuracy from 0.763830352735695 to 0.7716520484489502
Non decreasing pruning accuracy from 0.7716520484489502 to 0.8019136575919809
Non decreasing pruning accuracy from 0.8019136575919809 to 0.8024831985419751
Non decreasing pruning accuracy from 0.8024831985419751 to 0.8109503739985572
Non decreasing pruning accuracy from 0.8109503739985572 to 0.8211261723051221
Non decreasing pruning accuracy from 0.8211261723051221 to 0.8221893154117781
Non decreasing pruning accuracy from 0.8221893154117781 to 0.8235941830884307
Non decreasing pruning accuracy from 0.8235941830884307 to 0.8237840300717622
Pruned nodeId = 215
Pruning iteration 12 completed
Base accuracy = 0.8237840300717622
Non decreasing pruning accuracy from 0.0 to 0.7133310551695333
Non decreasing pruning accuracy from 0.7133310551695333 to 0.763830352735695
Non decreasing pruning accuracy from 0.763830352735695 to 0.7716520484489502
Non decreasing pruning accuracy from 0.7716520484489502 to 0.8019136575919809
Non decreasing pruning accuracy from 0.8019136575919809 to 0.8024831985419751
Non decreasing pruning accuracy from 0.8024831985419751 to 0.8109503739985572
Non decreasing pruning accuracy from 0.8109503739985572 to 0.8211261723051221
Non decreasing pruning accuracy from 0.8211261723051221 to 0.8221893154117781
Non decreasing pruning accuracy from 0.8221893154117781 to 0.8235941830884307
Non decreasing pruning accuracy from 0.8235941830884307 to 0.8237840300717622
Pruned nodeId = 210
Pruning iteration 13 completed
Base accuracy = 0.8237840300717622
Non decreasing pruning accuracy from 0.0 to 0.7133310551695333
Non decreasing pruning accuracy from 0.7133310551695333 to 0.763830352735695
Non decreasing pruning accuracy from 0.763830352735695 to 0.7716520484489502

Non decreasing pruning accuracy from 0.7716520484489502 to 0.8019136575919809
Non decreasing pruning accuracy from 0.8019136575919809 to 0.8024831985419751
Non decreasing pruning accuracy from 0.8024831985419751 to 0.8109503739985572
Non decreasing pruning accuracy from 0.8109503739985572 to 0.8211261723051221
Non decreasing pruning accuracy from 0.8211261723051221 to 0.8221893154117781
Non decreasing pruning accuracy from 0.8221893154117781 to 0.8235941830884307
Non decreasing pruning accuracy from 0.8235941830884307 to 0.8237080912784296
Non decreasing pruning accuracy from 0.8237080912784296 to 0.8237840300717622
Pruned nodeId = 196
Pruning iteration 14 completed
Base accuracy = 0.8237840300717622
Non decreasing pruning accuracy from 0.0 to 0.7133310551695333
Non decreasing pruning accuracy from 0.7133310551695333 to 0.763830352735695
Non decreasing pruning accuracy from 0.763830352735695 to 0.7716520484489502
Non decreasing pruning accuracy from 0.7716520484489502 to 0.8019136575919809
Non decreasing pruning accuracy from 0.8019136575919809 to 0.8024831985419751
Non decreasing pruning accuracy from 0.8024831985419751 to 0.8109503739985572
Non decreasing pruning accuracy from 0.8109503739985572 to 0.8211261723051221
Non decreasing pruning accuracy from 0.8211261723051221 to 0.8221893154117781
Non decreasing pruning accuracy from 0.8221893154117781 to 0.8235941830884307
Non decreasing pruning accuracy from 0.8235941830884307 to 0.8237080912784296
Non decreasing pruning accuracy from 0.8237080912784296 to 0.8237840300717622
Pruned nodeId = 172
Pruning iteration 15 completed
Base accuracy = 0.8237840300717622
Non decreasing pruning accuracy from 0.0 to 0.7133310551695333
Non decreasing pruning accuracy from 0.7133310551695333 to 0.763830352735695
Non decreasing pruning accuracy from 0.763830352735695 to 0.7716520484489502
Non decreasing pruning accuracy from 0.7716520484489502 to 0.8019136575919809
Non decreasing pruning accuracy from 0.8019136575919809 to 0.8024831985419751
Non decreasing pruning accuracy from 0.8024831985419751 to 0.8109503739985572
Non decreasing pruning accuracy from 0.8109503739985572 to 0.8211261723051221
Non decreasing pruning accuracy from 0.8211261723051221 to 0.8221893154117781
Non decreasing pruning accuracy from 0.8221893154117781 to 0.8235941830884307
Non decreasing pruning accuracy from 0.8235941830884307 to 0.8237080912784296
Non decreasing pruning accuracy from 0.8237080912784296 to 0.8237840300717622
Pruned nodeId = 163
Pruning iteration 16 completed
Base accuracy = 0.8237840300717622
Non decreasing pruning accuracy from 0.0 to 0.7133310551695333
Non decreasing pruning accuracy from 0.7133310551695333 to 0.763830352735695
Non decreasing pruning accuracy from 0.763830352735695 to 0.7716520484489502
Non decreasing pruning accuracy from 0.7716520484489502 to 0.8019136575919809
Non decreasing pruning accuracy from 0.8019136575919809 to 0.8024831985419751
Non decreasing pruning accuracy from 0.8024831985419751 to 0.8109503739985572
Non decreasing pruning accuracy from 0.8109503739985572 to 0.8211261723051221
Non decreasing pruning accuracy from 0.8211261723051221 to 0.8221893154117781
Non decreasing pruning accuracy from 0.8221893154117781 to 0.8235941830884307
Non decreasing pruning accuracy from 0.8235941830884307 to 0.8237080912784296
Non decreasing pruning accuracy from 0.8237080912784296 to 0.8237460606750959
Non decreasing pruning accuracy from 0.8237460606750959 to 0.8237840300717622
Pruned nodeId = 136
Pruning iteration 17 completed
Base accuracy = 0.8237840300717622
Non decreasing pruning accuracy from 0.0 to 0.7133310551695333
Non decreasing pruning accuracy from 0.7133310551695333 to 0.763830352735695
Non decreasing pruning accuracy from 0.763830352735695 to 0.7716520484489502

Non decreasing pruning accuracy from 0.7716520484489502 to 0.8019136575919809
Non decreasing pruning accuracy from 0.8019136575919809 to 0.8024831985419751
Non decreasing pruning accuracy from 0.8024831985419751 to 0.8109503739985572
Non decreasing pruning accuracy from 0.8109503739985572 to 0.8211261723051221
Non decreasing pruning accuracy from 0.8211261723051221 to 0.8221893154117781
Non decreasing pruning accuracy from 0.8221893154117781 to 0.8235941830884307
Non decreasing pruning accuracy from 0.8235941830884307 to 0.8237080912784296
Non decreasing pruning accuracy from 0.8237080912784296 to 0.8237460606750959
Non decreasing pruning accuracy from 0.8237460606750959 to 0.8237840300717622
Pruned nodeId = 120
Pruning iteration 18 completed
Base accuracy = 0.8237840300717622
Non decreasing pruning accuracy from 0.0 to 0.7133310551695333
Non decreasing pruning accuracy from 0.7133310551695333 to 0.763830352735695
Non decreasing pruning accuracy from 0.763830352735695 to 0.7716520484489502
Non decreasing pruning accuracy from 0.7716520484489502 to 0.8019136575919809
Non decreasing pruning accuracy from 0.8019136575919809 to 0.8024831985419751
Non decreasing pruning accuracy from 0.8024831985419751 to 0.8109503739985572
Non decreasing pruning accuracy from 0.8109503739985572 to 0.8211261723051221
Non decreasing pruning accuracy from 0.8211261723051221 to 0.8221893154117781
Non decreasing pruning accuracy from 0.8221893154117781 to 0.8235941830884307
Non decreasing pruning accuracy from 0.8235941830884307 to 0.8237080912784296
Non decreasing pruning accuracy from 0.8237080912784296 to 0.8237460606750959
No optimal pruning found!
Reduced Error Pruning finished
No of nodes after R.E.P. 97

Pruning finished.

Metrics after pruning
Accuracy = 0.8228727645517713 and Depth = 12
Precision = 0.3578615111461827 and Depth = 12
Recall   = 0.5626550868486352 and Depth = 12
The total number of nodes in the tree : 97

Collecting Data for different heights
Collecting Data for depth = 1
Accuracy = 0.8775866651478909 and Depth = 1
Precision = 0 and Depth = 1
Recall   = 0.0 and Depth = 1
The total number of nodes in the tree : 3
Collecting Data for depth = 2
Accuracy = 0.8775866651478909 and Depth = 2
Precision = 0 and Depth = 2
Recall   = 0.0 and Depth = 2
The total number of nodes in the tree : 5
Collecting Data for depth = 3
Accuracy = 0.7134449633595322 and Depth = 3
Precision = 0.2844320335095243 and Depth = 3
Recall   = 0.8846153846153846 and Depth = 3
The total number of nodes in the tree : 9
Collecting Data for depth = 4
Accuracy = 0.7646277100656871 and Depth = 4
Precision = 0.31408573928258965 and Depth = 4
Recall   = 0.7794665012406948 and Depth = 4
The total number of nodes in the tree : 15
Collecting Data for depth = 5

Accuracy = 0.7646277100656871 and Depth = 5
Precision = 0.31408573928258965 and Depth = 5
Recall   = 0.7794665012406948 and Depth = 5
The total number of nodes in the tree : 27
Collecting Data for depth = 6
Accuracy = 0.7823214489121768 and Depth = 6
Precision = 0.3252542136787853 and Depth = 6
Recall   = 0.7242555831265509 and Depth = 6
The total number of nodes in the tree : 47
Collecting Data for depth = 7
Accuracy = 0.7822455101188442 and Depth = 7
Precision = 0.3241350329177756 and Depth = 7
Recall   = 0.717741935483871 and Depth = 7
The total number of nodes in the tree : 78
Collecting Data for depth = 8
Accuracy = 0.7818658161521813 and Depth = 8
Precision = 0.32461388618338666 and Depth = 8
Recall   = 0.7236352357320099 and Depth = 8
The total number of nodes in the tree : 117
Collecting Data for depth = 9
Accuracy = 0.7804609484755287 and Depth = 9
Precision = 0.32255826859045506 and Depth = 9
Recall   = 0.7211538461538461 and Depth = 9
The total number of nodes in the tree : 175
Collecting Data for depth = 10
Accuracy = 0.7909784713520902 and Depth = 10
Precision = 0.3295980875541611 and Depth = 10
Recall   = 0.68424317617866 and Depth = 10
The total number of nodes in the tree : 251
Collecting Data for depth = 11
Accuracy = 0.7936363291187303 and Depth = 11
Precision = 0.33119560238204304 and Depth = 11
Recall   = 0.6727667493796526 and Depth = 11
The total number of nodes in the tree : 346
Collecting Data for depth = 12
Accuracy = 0.7932186657554011 and Depth = 12
Precision = 0.3285493827160494 and Depth = 12
Recall   = 0.6603598014888338 and Depth = 12
The total number of nodes in the tree : 440
Collecting Data for depth = 13
Accuracy = 0.7949652580020503 and Depth = 13
Precision = 0.33163107397090685 and Depth = 13
Recall   = 0.6647022332506204 and Depth = 13
The total number of nodes in the tree : 538
Collecting Data for depth = 14
Accuracy = 0.795458860158712 and Depth = 14
Precision = 0.32911992415863484 and Depth = 14
Recall   = 0.6460918114143921 and Depth = 14
The total number of nodes in the tree : 644
Collecting Data for depth = 15
Accuracy = 0.796901697232031 and Depth = 15
Precision = 0.3307851568721134 and Depth = 15
Recall   = 0.6442307692307693 and Depth = 15
The total number of nodes in the tree : 755
Collecting Data for depth = 16
Accuracy = 0.7942818088620572 and Depth = 16
Precision = 0.32697160883280757 and Depth = 16

Recall = 0.6429900744416873 and Depth = 16
The total number of nodes in the tree : 871
Collecting Data for depth = 17
Accuracy = 0.7967877890420321 and Depth = 17
Precision = 0.32838709677419353 and Depth = 17
Recall = 0.6315136476426799 and Depth = 17
The total number of nodes in the tree : 993
Collecting Data for depth = 18
Accuracy = 0.7958765235220412 and Depth = 18
Precision = 0.32331691297208537 and Depth = 18
Recall = 0.6107320099255583 and Depth = 18
The total number of nodes in the tree : 1105
Collecting Data for depth = 19
Accuracy = 0.7995975243953374 and Depth = 19
Precision = 0.32408359027064065 and Depth = 19
Recall = 0.586848635235732 and Depth = 19
The total number of nodes in the tree : 1215
----------Task-3-Completed----------

Discussion

At the start we were stuck with the 87.5% Accuracy constant decision tree (Always 0 prediction), but by implementing the best option method of decision node class we managed to get an Recall of 67% with precision of 34% and accuracy 80.24%.

Then we compared and selected the best tree from trees built using 10 random data splits.
We then ran a reduced error pruning algorithm over the best tree and achieved a 2% gain in accuracy and reduced the complexity of the tree from 447 nodes to 97 nodes. We printed both the unpruned and pruned version of the tree.

We plotted data for variation in performance with pruning and variation of performance with different depths. And for the accuracy vs height we also noticed that the improvement was getting less significant as we increased the depth of the tree indicating that the tree has now started to overfit the data around the depth = 12 to 15.

Bayesian (Naïve Bayes) Classifier

Task 1

To split the dataset and identify categorical and continuous attributes, we use the same functions as in Task 1 of Decision trees.

```
def train_test_split(df,train_sample=0.5,target_col= LABEL)
```

Observing the number of unique values and the data type of the values in a feature we decide if it is a categorical string type feature. We find that the following features are of the said type:

- Gender
- Vehical_age
- Vehical_damage

The above attributes will need to be encoded with integers. We hard-code the encoding and make necessary changes in the dataset.

```
vehicle_age_dict = {
    '< 1 Year' : 0,
    '1-2 Year' : 1,
    '> 2 Years': 2
}
df["Vehicle_Age_ordinal"] = df.Vehicle_Age.map(vehicle_age_dict)
gender_dict = {
    'Male': 0,
    'Female': 1
}
df["Gender_encoded"] = df.Gender.map(gender_dict)
vehicle_damage_dict = {
    'No' : 0,
    'Yes' : 1
}
df["Vehicle_Damage_encoded"] = df.Vehicle_Damage.map(vehicle_damage_dict)
```

Task 2

For each feature, we calculate the bounds for potential outliers and filter out the data points lying outside the bounds for those particular features.

Task 2 Output

```
----------Task-2-Started----------
Initial Dataset count 131689

Feature Age
Mean            :38.8106068084654
Standard Deviation :15.494455932192853
Lower cutoff = -7.672760988113161
Upper cutoff = 85.29397460504396
Outlier count = 0
Filtering the data set

Feature Driving_License
Mean            :0.9979193402638034
Standard Deviation :0.04556694370158682
Lower cutoff = 0.8612185091590429
Upper cutoff = 1.1346201713685637
Outlier count = 274
Filtering the data set

Feature Region_Code
Mean            :26.407731233116465
Standard Deviation :13.222132003580189
Lower cutoff = -13.258664777624105
Upper cutoff = 66.07412724385703
Outlier count = 0
Filtering the data set

Feature Previously_Insured
Mean            :0.45802229578054254
Standard Deviation :0.4982366519278903
Lower cutoff = -1.0366876600031283
Upper cutoff = 1.9527322515642136
Outlier count = 0
Filtering the data set

Feature Annual_Premium
Mean            :30472.05259673553
Standard Deviation :17120.169982518542
Lower cutoff = -20888.457350820096
Upper cutoff = 81832.56254429116
Outlier count = 690
Filtering the data set
```

Feature Policy_Sales_Channel
Mean :112.37389175750621
Standard Deviation :54.06433622706852
Lower cutoff = -49.81911692369937
Upper cutoff = 274.5669004387118
Outlier count = 0
Filtering the data set

Feature Vintage
Mean :154.47036144578314
Standard Deviation :83.79068007173944
Lower cutoff = -96.9016787694352
Upper cutoff = 405.8424016610015
Outlier count = 0
Filtering the data set

Feature Vehicle_Age_ordinal
Mean :0.6073360107095047
Standard Deviation :0.5672619232019509
Lower cutoff = -1.094449758896348
Upper cutoff = 2.3091217803153574
Outlier count = 0
Filtering the data set

Feature Gender_encoded
Mean :0.4607917383820998
Standard Deviation :0.49846224820670737
Lower cutoff = -1.0345950062380223
Upper cutoff = 1.956178483002222
Outlier count = 0
Filtering the data set

Feature Vehicle_Damage_encoded
Mean :0.5050908395486708
Standard Deviation :0.49997599500437545
Lower cutoff = -0.9948371454644556
Upper cutoff = 2.0050188245617973
Outlier count = 0
Filtering the data set
Final dataset count after removing outliers = 130725

We drop the feature columns for which all the data points have one unique value.

Removing feature Driving_License, since the unique value left in the samples = 1

We get the normalized dataset using min-max normalization.

```
normalized_df=(df-df.min())/(df.max()-df.min())
```

Task 3

The following functions are used to implement the Naive Bayes classifier for a given training set and data set.

```
def calculate_prior(train_data, label): # Calculating Prior for all classes of label

def calculate_likelihood_categorical(summary, feature, feature_value, label, Y, laplaceCorrection =
    False, uniqueFeatureCount = 0):
# Calculating P(X = x | Label = Y) for categorical features


def calculate_likelihood_gaussian(summary, feature, feature_value, label, Y):
# Calculating P(X = x  | label = Y) for continuous features



# class to feature to summary
'''
summary[feature][class][mean / std] for continuous
summary[feature][class][feature_value] = (tuple) (count of feature=feature value | class , count of feature
    class)
The second one is count so that we may use the same summary for Laplace correction
'''
def generateSummary(train_data, label):
#function to pre compute the summary for each label class for each feature

def naive_bayes_algo(train_data, test_X, label, laplaceCorrection = False):
#computes the list of predictions for each datapoint in the test set

def GetAccuracy(Y_test, Y_pred, printDetails = True):
# Evaluates Accuracy


def GetPrecision(Y_test, Y_pred, printDetails = True):
# Evaluates precision for predicting 1

def GetRecall(Y_test, Y_pred, printDetails = True):
# Evaluates recall for predicting 1


def evaluateAlgo(Y_test, Y_pred, printDetails = True):
# Function to evaluate accuracy, recall and precision

def KFolds(dataframe, k = 10): #returns k split blocks of the dataset
```

```
def kFoldValidation(dataframe, algorithm, label, k=10, laplaceCorrection = False, _folds = None):
#creates training set and test set with the k data splits, runs the model on each set and measures
  accuracy with provision to apply laplace correction
```

Task 3 output

```
10 fold validation started
Iter : 1  started
Accuracy  = 0.7675361431958999
Precision = 0.28579970104633784
Recall    = 0.5952677459526775
Iter : 2  started
Accuracy  = 0.7666947142966419
Precision = 0.28345534407027817
Recall    = 0.6161680458306811
Iter : 3  started
Accuracy  = 0.7640174405262755
Precision = 0.283907031479847
Recall    = 0.5971534653465347
Iter : 4  started
Accuracy  = 0.760881205538132
Precision = 0.2932636469221835
Recall    = 0.5934195064629847
Iter : 5  started
Accuracy  = 0.7694484816033045
Precision = 0.283979863784424
Recall    = 0.6167202572347267
Iter : 6  started
Accuracy  = 0.7653943241796068
Precision = 0.2844063792085056
Recall    = 0.5992532669570628
Iter : 7  started
Accuracy  = 0.7601162701751702
Precision = 0.2779394644935972
Recall    = 0.593167701863354
Iter : 8  started
Accuracy  = 0.7661592595425687
Precision = 0.28441710603397774
Recall    = 0.6126182965299685
Iter : 9  started
Accuracy  = 0.7674596496596038
Precision = 0.28336325553560743
Recall    = 0.5948492462311558
Iter : 10  started
Accuracy  = 0.7705846342209979
Precision = 0.28299607369374813
Recall    = 0.6002562459961563

Mean accuracy = 0.7658292122938202
Mean precision = 0.28435278662685065
Mean recall = 0.6018873778405303
10 fold validation finished
----------Task-3-Completed----------
```

Task 4

We apply kFoldsValidation with laplaceCorrection parameter set true.

Task 4 Output

```
----------Task-4-Started----------
10 fold validation started
Iter : 1  started
Accuracy  = 0.7677656238047885
Precision = 0.2856714628297362
Recall    = 0.5933997509339975
Iter : 2  started
Accuracy  = 0.7668477013692343
Precision = 0.28311306901615274
Recall    = 0.6136218968809676
Iter : 3  started
Accuracy  = 0.7641704275988679
Precision = 0.28394698085419734
Recall    = 0.5965346534653465
Iter : 4  started
Accuracy  = 0.760881205538132
Precision = 0.2930232558139535
Recall    = 0.5922444183313749
Iter : 5  started
Accuracy  = 0.7696014686758968
Precision = 0.284020160094871
Recall    = 0.6160771704180065
Iter : 6  started
Accuracy  = 0.7656238047884953
Precision = 0.28465858705291164
Recall    = 0.5992532669570628
Iter : 7  started
Accuracy  = 0.7603457507840587
Precision = 0.2780530457592539
Recall    = 0.5925465838509317
Iter : 8  started
Accuracy  = 0.7661592595425687
Precision = 0.2841642228739003
Recall    = 0.6113564668769716
Iter : 9  started
Accuracy  = 0.7675361431958999
Precision = 0.2830584707646177
Recall    = 0.592964824120603
Iter : 10  started
Accuracy  = 0.77089072543618
Precision = 0.2832072617246596
Recall    = 0.5996156310057655

Mean accuracy = 0.7659822110734122
Mean precision = 0.2842916516784254
Mean recall = 0.6007614662841028
10 fold validation finished
----------Task-4-Finished----------
```

Discussion:

We built a naive bayes classifier that can handle categorical as well as continuous features. For categorical features we also added the laplace correction to ensure that the likelihood always remains non zero, contributing to the smoothing of the classifier. Finally we achieved a mean Accuracy of 76.59% with 28% precision and 60% recall.

Assignment Discussion

On comparing both the classifiers, the decision tree performed better but the implementation was much more simpler for the naive bayes classifier. Thus each of them have their own benefits.

The Naive Bayes classifier can also provide us a relative confidence of predicting the class by simply returning the relative ratio of posterior probabilities.

The decision tree provides a more comprehensible model, which can be easily visualized. This lets us analyze the model's decision. Once the tree is built, the prediction performance is very fast since it only needs to traverse the depth of the tree.