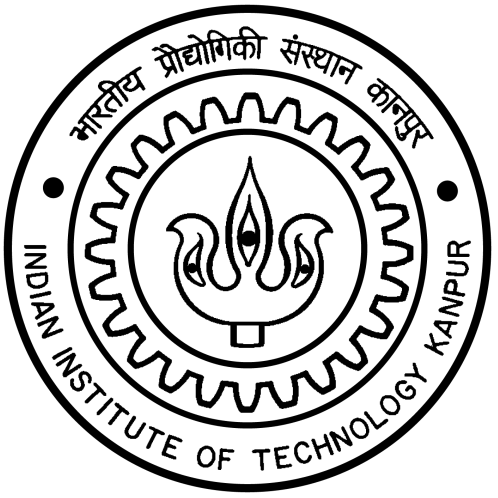
MTH552 PROJECT REPORT

(Instructor- Dr. Amit Mitra)



Implementation of Decision Tree and Random Forest, and Analysis on numerous data sets

Project undertaken by :

● Aniket Kumar (190134), Department of Mathematics and Scientific Computing
● Munish Gupta (190519), Department of Mathematics and Scientific Computing

Abstract

Decision tree is one of the most powerful and popular tool for classification. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each terminal holds a class label, but these decision trees are highly influenced by the training data sets and even some outliers can make the tree predict inaccurate results. Random forest overcomes this weakness by creating numerous decision trees which train on a random subset of data and using random subset of features for training. This randomness assures that the model is not over-fitting to the data. In this project we implemented the decision tree and random forest from scratch and want to see their performance on various data sets. Different impurity metrics such as gini index, entropy and misclassification error rate are also implemented to get a better in-depth comparison. Data sets used in this project are Car Evaluation, Iris, Breast Cancer Wisconsin (Diagnostic), E-coli dataset and Number Recognition dataset.

Acknowledgement

I take this opportunity to express my deepest sense of gratitude and sincere thanks to everyone who helped me to complete this work successfully. I express my sincere thanks to Dr.Amit Mitra, Department of Mathematics and Statistics, for providing me with all the necessary facilities, support and guidance throughout the course and project timeline. Our journey of accomplishing the project really involves many ones to whom we are highly obliged. Thanks to Open Source dataset repository - UCL, for providing dataset for testing our implementation.

Content

Introduction

In most real life datasets, simple supervised or unsupervised non-parametric approaches, such as decision tree and k-nearest neighbours mostly perform similar, or comparable to new SOTA methods and algorithms. The reason and explanation being in their simplicity and their similarity to the decision making process in real life.
A decision tree is essentially a chart or tree-like structure with each node either being an internal node (where the data points are split based on a certain feature and threshold) or a terminal node(all data points that end up here are classified into one of the classes).
In this project, we implement Decision Tree from scratch using numpy arrays and pandas dataframe. But, as the data features increase, decision trees tend to run into problems of high variance, i.e., they overfit to the current data and thus aren't able to generalise well when tested over similar or other relevant data. This problem is solved by Random Forest, viz, a collection of trees made by randomly choosing features and random data points from the original data set. We also use various popular metrics(entropy, misclassification error rate and Gini Index) to check accuracy and performance of our implementation on popular datasets.

Decision Tree

Decision tree is a supervised non parametric method of learning that is used both for classification and regression. In this project we will only be using it for classification. Decision tree comprises internal nodes and terminal nodes, with each internal there is a left child, right child, a feature and a threshold is attached that will be used to test the feature factor in classifying it. With each terminal we have a class label, if a vector reaches this node that label will be assigned to it.

Implementation

- **Node**

Each node is either an internal or a terminal node, having left child, right child, feature and threshold attributes in case of internal node and only label attribute in case of terminal node (if the label of any Node object is not None, it is terminal node).

```
class Node:
    def __init__(self, feature=None, threshold=None, left=None, right=None,label=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.label = label

    def is_terminal(self):
        return self.label is not None
```

- **Intialisation**

We will be initializing a decision tree with-

- max depth attribute, maximum depth a node can have before being classified as a terminal node.
- Minimum sample split attribute, minimum sample the node have must have in order to continue splitting
- Scoring metric, which metric out of gini index, entropy and misclassification should be used for splitting

```
class DecisionTree:
    def __init__(self, max_depth=100, min_samples_split=2,scoring='entropy'):
      self.max_depth = max_depth
      self.min_samples_split = min_samples_split
      self.scoring=scoring
      self.root = None
```

- **Creating best split**

Best feature and best threshold is calculated by traversing over every feature and every threshold, and selecting the best ones which have the highest goodness score.

```
for feat in features:
  X_feat = X[:, feat]
  thresholds = np.unique(X_feat)
  for thresh in thresholds:
    score = self.goodness(X_feat, y, thresh)
    if score > split['score']:
      split['score'] = score
      split['feat'] = feat
      split['thresh'] = thresh
return split['feat'], split['thresh']
```

- **Splitting the Node into according to the best split**

The data is then split for the children nodes according to the best split parameters.

```
def create_split(self, X, thresh):
    l_idx = np.argwhere(X <= thresh).flatten()
    r_idx = np.argwhere(X > thresh).flatten()
    return l_idx, r_idx
```

- **Terminal Nodes**

A node will be classified as terminal when the depth of the node is equal to the max depth or no. of data points in the node is less than the minimum split size or the data belonging to that is of a single class.

```
def is_finished(self, depth):
    if (depth >= self.max_depth or self.n_class_labels
        == 1 or self.n_samples < self.min_samples_split):
        return True
    return False
```

- **Fitting the data**

The data is fitted to the tree by building the tree from the root recursively, with help of the below functions

```
def build_tree(self, X, y, depth=0,forest=0):
  self.n_samples, self.n_features = X.shape
  self.n_class_labels = len(np.unique(y))
  if forest is 1:### for random forest
    self.feats=[]
    for i in range(self.n_features):
      if(np.random.randint(2)):
        self.feats.append(i)
    if(len(self.feats)==0):
      self.feats=range(self.n_features)
  else:
    self.feats=range(self.n_features)
  # stopping criteria
  if self.is_finished(depth):
    most_common_Label = np.argmax(np.bincount(y))
    return Node(label=most_common_Label)
  #get best split
  best_feat, best_thresh = self.best_split(X, y, self.feats)

  # grow children recursively
  l_idx, r_idx = self.create_split(X[:, best_feat], best_thresh)
  if(len(r_idx) is 0 or len(l_idx) is 0):
    return Node(label=np.argmax(np.bincount(y)))
  else :
    left_child = self.build_tree(X[l_idx, :], y[l_idx], depth + 1)
    right_child = self.build_tree(X[r_idx, :], y[r_idx], depth + 1)
    return Node(best_feat, best_thresh, left_child, right_child)
```

- **Predicting the data**

The data is traversed through the tree till it reaches a terminal and then the label associated with that terminal node is returned as the predicted class.

```
def traverse(self, x, node):
  if node.is_terminal():
    return node.label
  if x[node.feature] <= node.threshold:
    return self.traverse(x, node.left)
  return self.traverse(x, node.right)

def predict(self, X):
  predictions = [self.traverse(x, self.root) for x in X]
  return np.array(predictions)
```

Random Forest

Random forest is also a supervised, non parametric method of learning, just like its name suggests it comprises a number of decision trees, it overcomes the biggest weakness of the decision tree which is overfitting to the training data. Outliers influence the decision tree a lot. The Random forest overcomes this weakness by creating numerous decision trees and training them on a random subset of the data, with random features. Then the majority voting rule is used to predict the outcome, this randomness induced by the data and features help in reducing the problem of overfitting.

Implementation

Initialisation

We will be initializing a decision tree with are-

- Size of the random forest i.e no. of decision trees to be created in the forest
- Batch size, it is the size of the data on which each decision tree should, this data is generated through bagging.
- max depth attribute, maximum depth a node can have before being classified as a terminal node in underlying decision trees

- Minimum sample split attribute, minimum sample the node have must have in order to continue splitting in the underlying decision trees
- Scoring metric, which metric out of gini index, entropy and misclassification should be used for splitting in the underlying decision trees.

```
def __init__(self, max_depth=100, min_samples_split=2,size=100,scoring='entropy',batch_size=256):
  self.max_depth = max_depth
  self.min_samples_split = min_samples_split
  self.size=size
  self.root = None
  self.scoring=scoring
  self.batch size=batch size
```

- **Random Subset of data**

Random subset of the training is generated through bagging, uniform sampling is done from the given training data set with replacement, samples equal to the batch size is generated for each tree and these trees are then trained on them.

```
z=np.random.choice(X.shape[0],int(self.batch_size),replace=True)
tree=DecisionTree(max_depth=self.max_depth,min_samples_split=self.min_samples_split,scoring=self.scoring)
tree.fit(X[z,:],y[z],forest=1)
```

- **Random Features**

Each feature has a probability of 0.5 to be selected by a given decision tree in the random forest. Thus making the feature set of each decision tree as random.

```
self.feats=[]
for i in range(self.n_features):
  if(np.random.randint(2)):
    self.feats.append(i)
if(len(self.feats)==0):
  self.feats=range(self.n_features)
```

- **Prediction**

The test vector is traversed through every decision tree and majority voting rule is applied on the outcome of the decision trees.

```
def predict(self,X):
  pred=np.zeros((X.shape[0],self.classes))
  for i in range(self.size):
    y=self.trees[i].predict(X)
    for j in range(len(y)):
      pred[j,y[j]]=pred[j,y[j]]+1
  pred_ = np.argmax(pred,axis=1)
  return pred_
```

Impurity Metrics

These three impurity metrics are used in our implementation of decision trees:

- **Gini Index-**

  $$Gini = 1 - \sum_{i=1}^{C} p_i^2$$

  , where p_i is the probability of a datapoint being in ith class. It is certainly very simple to calculate and highly efficient. It is the most preferred metric for decision trees.

- **Entropy-**

  $$\sum_{i=1}^{C} -p_i log p_i$$

  , where p_i is the probability of a datapoint being in ith class. Entropy is not preferred due to the 'log' function as it increases the computational complexity.

- **Misclassification Error rate-**

  It is also rather simple. As the name suggest it is $Mer = 1 - Accuracy$

  Since it is directly calculated from the data present, the chance of it being overfit to the data is higher. It also fails when the probability of two classes are really close.

Data Set Description

We have tested our implementation of decision tree and random forest on multiple famous datasets so as to ensure generality over different data types and ranges.

Car Evaluation Dataset [link]

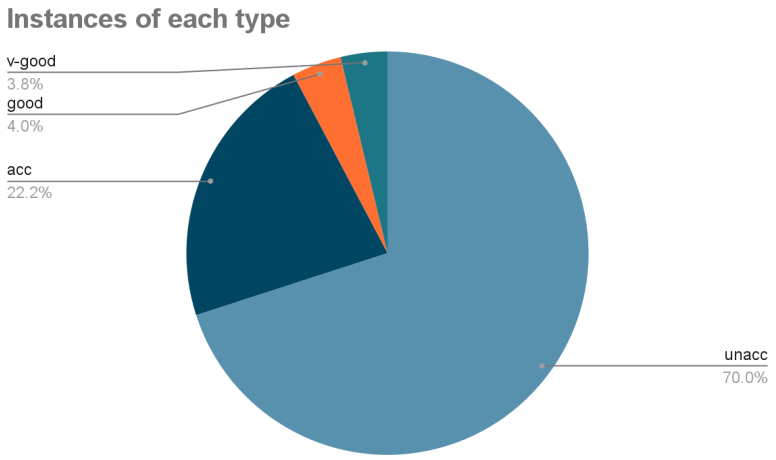
This relatively small dataset has 6 features (independent variables), all of them being ordinal. The predicted or dependent variable is the class rating of the car, being of type ordinal. Following is the description of dataset features and dependent variable:

buying v-high, high, med, low (buying price of the vehicle)
maint v-high, high, med, low (maintenance cost of the vehicle)
doors 2, 3, 4, 5-more (number of doors)
persons 2, 4, more (to accommodate number of people)
lug_boot small, med, big (luggage or boot space of the vehicle)
safety low, med, high (safety rating of the car)

class - unacc, acc, good, vgood (class rating of the vehicle)

The dataset has 1728 instances and after performing relevant checks, we've found no missing values.
String type of ordinal features was given integral values, with greater number meaning better values for ordinal types. The dataset was randomly

distributed into test and training data in 3:7 ratio.



Above is the visualisation of percent of data points present in each class of predicted variable.

Accuracy achieved using Decision Tree and Random Forest using different impurity metrics

Decision Tree using Entropy : 98.07%
Decision Tree using Gini Index : 98.84%
Decision Tree using mer : 81.69%

Random Forest using Entropy : 97.49%
Random Forest using Gini Index : 97.68%
Random Forest using mer : 87.09%

Iris Dataset [link]

This highly famous dataset has 150 attributes, 50 of each class (Iris Setosa, Iris Versicolour, Iris Virginica). It has 4

features, following is the description of them along with predicted feature:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:

-- Iris Setosa

-- Iris Versicolour

-- Iris Virginica

The dataset was randomly distributed into test and training data in 3:7 ratio.

Accuracy achieved using Decision Tree and Random Forest using different impurity metrics

Decision Tree using Entropy : 95.55%
Decision Tree using Gini Index : 95.55%
Decision Tree using mer : 95.55%

Random Forest using Entropy : 95.55%
Random Forest using Gini Index : 95.55%
Random Forest using mer : 95.55%

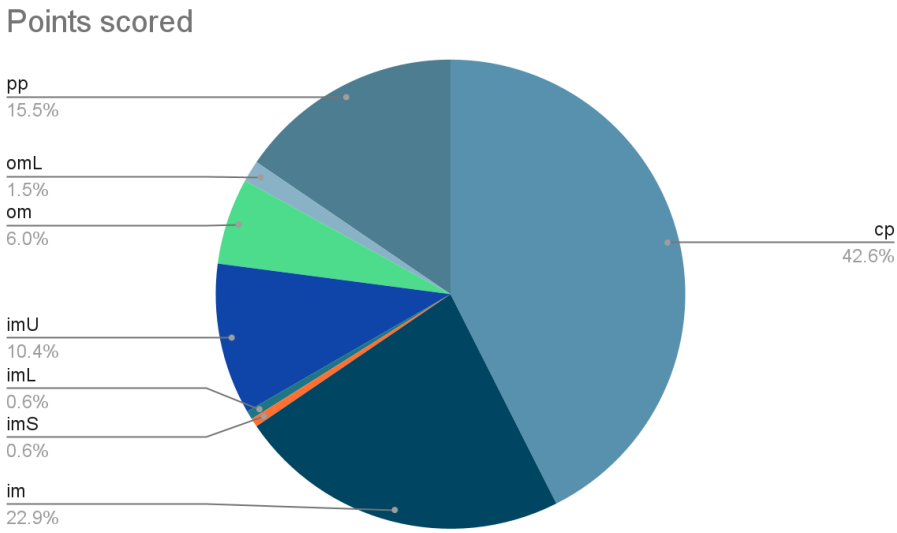
E-Coli Dataset [link]

Ecoli is another fairly famous data set consisting of 8 attributes and 336 data points. Following is the description of its attributes:

1. Sequence Name
2. mcg: McGeoch's method for signal sequence recognition.
3. gvh: von Heijne's method for signal sequence recognition.
4. lip: von Heijne's Signal Peptidase II consensus sequence score. Binary attribute.
5. chg: Presence of charge on N-terminus of predicted lipoproteins. Binary attribute.
6. aac: score of discriminant analysis of the amino acid content of outer membrane and periplasmic proteins.
7. alm1: score of the ALOM membrane spanning region prediction program.
8. alm2: score of ALOM program after excluding putative cleavable signal regions from the sequence.

The dataset was randomly distributed into test and training data in 3:7 ratio.

We convert the ordinal response variable. Now, I visualize our data wrt the response variable.



Accuracy achieved using Decision Tree and Random Forest using different impurity metrics

Decision Tree using Entropy : 83.16%
Decision Tree using Gini Index : 83.16%
Decision Tree using mer : 82.17%

Random Forest using Entropy : 85.14%
Random Forest using Gini Index : 84.15%
Random Forest using mer : 84.15%

Breast Cancer Wisconsin (Diagnostic) dataset[link]

The dataset consists of 30 predictors & a binary dependent variable labeled as diagnosis. The response variable takes the letter ‘M’ denoting the tumour is Malignant and letter ‘B’ denoting the tumour is benign.

The features are as follows:

feature1- id

feature2- **diagnosis**

feature3- radius_mean (mean of distances from center to points on the

perimeter)

feature4- texture_mean (standard deviation of gray-scale values)

feature5- perimeter_mean

feature6- area_mean

feature7- smoothness_mean (local variation in radius lengths)

feature8- compactness_mean (perimeter^2 / area - 1.0)

feature9- concavity_mean (severity of concave portions of the contour)

feature10- concave points_mean (number of concave portions of the

contour)

feature11- symmetry_mean

feature12- fractal_dimension_mean("coastline approximation"-1)

feature13- radius_se

feature14- texture_se

feature15- area_se

feature16- smoothness_se

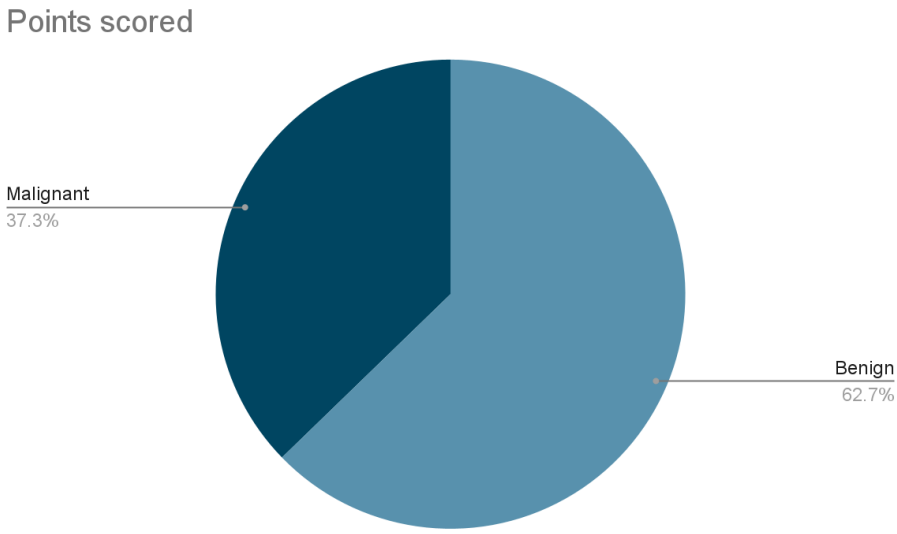
feature17- compactness_se

feature18- concavity_se

feature19- concave points_se
feature20- symmetry_se
feature21- fractal_dimension_se
feature22- perimeter_se
feature23- radius_worst
feature24- texture_worst
feature25- perimeter_worst
feature26- area_worst
feature27- smoothness_worst
feature28- compactness_worst
feature29- concavity_worst
feature30- concave points_worst
feature31- symmetry_worst
feature32- fractal_dimension_worst

Here, we have 569 data points and after missing value check, we conclude there is no missing value in the dataset.

We convert the binary response variable to 0 for Benign and 1 for malignant. Now, I visualize our data wrt the response variable (Diagnosis).



The dataset was randomly distributed into test and training data in 3:7 ratio.

Accuracy achieved using Decision Tree and Random Forest using different impurity metrics

Decision Tree using Entropy : 91.66%
Decision Tree using Gini Index : 90.78%
Decision Tree using mer : 91.22%

Random Forest using Entropy : 94.29%
Random Forest using Gini Index : 92.98%
Random Forest using mer : 93.42%

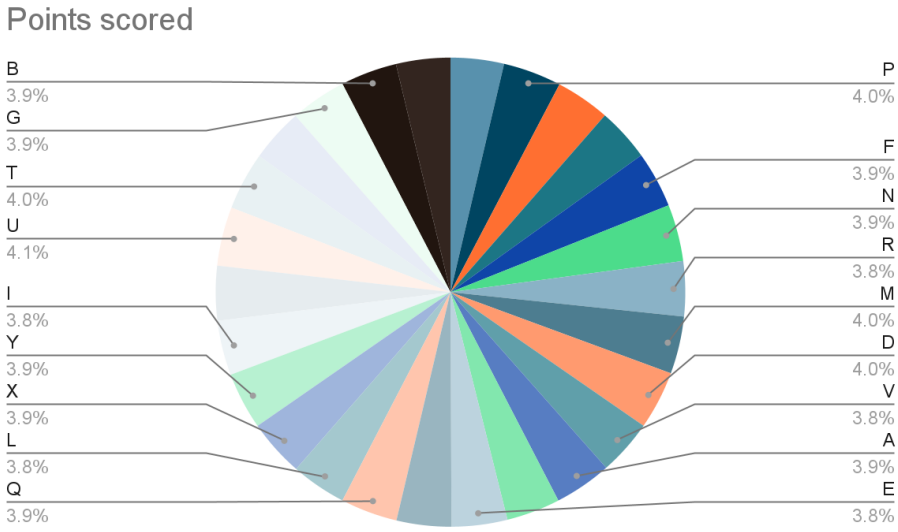
Number Recognition Dataset [link]

The dataset consists of 16 attributes and 20000 instances. The predicted variable is class of letter i.e. (A to Z) which is relabelled as 0 to 25 (not in the same order).
Following is the feature description for attributes:

1. lettr - capital letter (26 values from A to Z)
2. x-box - horizontal position of box (integer)
3. y-box - vertical position of box (integer)
4. width - width of box (integer)
5. high - height of box (integer)
6. onpix - total # on pixels (integer)
7. x-bar - mean x of on pixels in box (integer)
8. y-bar - mean y of on pixels in box (integer)
9. x2bar - mean x variance (integer)

10. y2bar - mean y variance (integer)
11. xybar - mean x y correlation (integer)
12. x2ybr - mean of x * x * y (integer)
13. xy2br - mean of x * y * y (integer)
14. x-ege - mean edge count left to right (integer)
15. xegvy - correlation of x-ege with y (integer)
16. y-ege - mean edge count bottom to top (integer)
17. yegvx - correlation of y-ege with x (integer)

After checking we found no missing value. Following is the proportion of digits:



Accuracy achieved using Decision Tree and Random Forest using different impurity metrics

Decision Tree using Entropy : 86.88%
Decision Tree using Gini Index : 81.51%

Decision Tree using mer : 81.91%

Random Forest using Entropy : 91.01%
Random Forest using Gini Index : 88.75%
Random Forest using mer : 86.38%

Result and Discussions

Both the models performed very well in all the data sets, having accuracy over 80 percent in all the data sets.
As the complexity of the data sets start to increase the Random Forest starts to take a slight leap ahead of the decision tree in terms of accuracy.
In simple datasets such as Iris(4 feature), Car evaluation(6 feature), which has only few features, the performance of both of them is identical, but with increase in the size of feature vector we get that random forest performed slightly better in every metric as seen in Breast Cancer data set(31 features) and Letter Recognition dataset(16 features)

References

- https://en.wikipedia.org/wiki/Random_forest
- https://www.ibm.com/cloud/learn/random-forest
- https://online.stat.psu.edu/stat508/lesson/11/11.2
- https://en.wikipedia.org/wiki/Decision_tree