**Lab Task: Classification through Random Forests**

|  |  |
| --- | --- |
| **Name** | **Malik Hassan Raza** |
| **Reg** | **208046** |
| **Class** | **BSCS 7B** |

In this lab, you are supposed to implement the Random Forest Classifier. Data in excel files (both the training and test sets) are uploaded on LMS. In the said training and test data files, each row contains data about one instance of a plant category where four predictors/attributes are recorded for each plant (namely, leaf length, leaf width, flower length, and flower width), while “plant” is the target class which could be any one of the following at a time: “Arctica” or “Harlequin” or “Caroliniana”.

You are supposed to develop random forests with

1. 100 trees
2. 300 trees
3. 500 trees, and

Figure out how the results vary on the test data (How results vary by increasing the number of trees). Please submit your results for the complete test set using the three abovementioned random forests (a, b, and c).

**Hint:** Since there are numerical predictors, you may like to look into binning/discretization. Furthermore, *Scikit-learn* might be a useful resource during implementation, feel free to use this-or any other suitable library.

**Happy coding.**

|  |
| --- |
| **Code** |
| import numpy as np from random import seed from random import randrange from csv import reader from math import sqrt  eps = np.finfo(float).eps # smallest possible number  # load csv files def load\_csv(filename):  dataset = list()  row = 1  with open(filename, 'r') as file:  csv\_reader = reader(file)  for row in csv\_reader:  if not row:  continue  dataset.append(row)  return dataset   # string to float converter def str2float(dataset, column):  for row in dataset:  row[column] = float(row[column].strip())   # string to integer converter def str2int(dataset, column):  class\_values = [row[column] for row in dataset]  unique = set(class\_values)  lookup = dict()  for i, value in enumerate(unique):  lookup[value] = i  for row in dataset:  row[column] = lookup[row[column]]  return lookup   # Split a dataset on attribute and attribute value def test\_split(index, value, dataset):  left, right = list(), list()  for row in dataset:  if row[index] < value:  left.append(row)  else:  right.append(row)  return left, right   # Calculate the Gini index for a split dataset def gini\_index(groups, classes):  n\_instances = float(sum([len(group) for group in groups]))  gini = 0.0  for group in groups:  size = float(len(group))  if size == 0:  continue  score = 0.0  for class\_val in classes:  p = [row[-1] for row in group].count(class\_val) / size  score += p \* p  gini += (1.0 - score) \* (size / n\_instances)  return gini   # Select the best split point def get\_split(dataset, n\_features):  class\_values = list(set(row[-1] for row in dataset))  b\_index, b\_value, b\_score, b\_groups = 999, 999, 999, None  features = list()  while len(features) < n\_features:  index = randrange(len(dataset[0]) - 1)  if index not in features:  features.append(index)  for index in features:  for row in dataset:  groups = test\_split(index, row[index], dataset)  gini = gini\_index(groups, class\_values)  if gini < b\_score:  b\_index, b\_value, b\_score, b\_groups = index, row[index], gini, groups  return {'index': b\_index, 'value': b\_value, 'groups': b\_groups}   # Create a terminal node value def to\_terminal(group):  outcomes = [row[-1] for row in group]  return max(set(outcomes), key=outcomes.count)   # child splits for a node or make terminal def split(node, max\_depth, min\_size, n\_features, depth):  left, right = node['groups']  del (node['groups'])  # no split check  if not left or not right:  node['left'] = node['right'] = to\_terminal(left + right)  return  # max depth check  if depth >= max\_depth:  node['left'], node['right'] = to\_terminal(left), to\_terminal(right)  return  # process left child  if len(left) <= min\_size:  node['left'] = to\_terminal(left)  else:  node['left'] = get\_split(left, n\_features)  split(node['left'], max\_depth, min\_size, n\_features, depth + 1)  # process right child  if len(right) <= min\_size:  node['right'] = to\_terminal(right)  else:  node['right'] = get\_split(right, n\_features)  split(node['right'], max\_depth, min\_size, n\_features, depth + 1)   # Building decision tree def build\_tree(train, max\_depth, min\_size, n\_features):  root = get\_split(train, n\_features)  split(root, max\_depth, min\_size, n\_features, 1)  return root   # Making prediction def predict(node, row):  if row[node['index']] < node['value']:  if isinstance(node['left'], dict):  return predict(node['left'], row)  else:  return node['left']  else:  if isinstance(node['right'], dict):  return predict(node['right'], row)  else:  return node['right']   # random subsample from the dataset with replacement def subsample(dataset, ratio):  sample = list()  n\_sample = round(len(dataset) \* ratio)  while len(sample) < n\_sample:  index = randrange(len(dataset))  sample.append(dataset[index])  return sample   # Make a prediction with a list of bagged trees def bagging\_predict(trees, row):  predictions = [predict(tree, row) for tree in trees]  return max(set(predictions), key=predictions.count)   # Random Forest Algorithm def random\_forest(train, test, max\_depth, min\_size, sample\_size, n\_trees, n\_features):  trees = list()  for i in range(n\_trees):  sample = subsample(train, sample\_size)  tree = build\_tree(sample, max\_depth, min\_size, n\_features)  trees.append(tree)  predictions = [bagging\_predict(trees, row) for row in test]  return (predictions)   # Test the random forest algorithm seed(2)  # loading the csv files into datasets training\_file = 'TrainingSet.csv' test\_file = 'TestSet1.csv'  training\_dataset = load\_csv(training\_file) test\_dataset = load\_csv(test\_file)  # removing the headings row training\_dataset.pop(0) test\_dataset.pop(0)  # convert string attributes to integers in training set for i in range(0, len(training\_dataset[0]) - 1):  str2float(training\_dataset, i)  print("Following lookup table indicates what integer represents each class according to my labelling:\n") # convert class col to int in training set print(str2int(training\_dataset, len(training\_dataset[0]) - 1)) # converting str attr to float in test set for i in range(0, len(test\_dataset[0]) - 1):  str2float(test\_dataset, i)   # actual task max\_depth = 10 min\_size = 1 sample\_size = 1.0 n\_features = int(sqrt(len(training\_dataset[0]) - 1))  for n\_trees in [100, 300, 500]:  predictions = random\_forest(training\_dataset, test\_dataset, max\_depth, min\_size, sample\_size, n\_trees, n\_features)  print(str(n\_trees) + ' Trees Predictions: %s' % predictions) |
| **SS** |
|  |