Final exam report of assignments

Subarna Khorshed

UG02-48-18-010

1 KNN:

The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems. It's easy to implement and understand, but has a major drawback of becoming significantly slows as the size of that data in use grows.

Algorithm Explanation:

• Take Data input from CSV file

• Decision tree is built as below- o Find which attribute has the maximum information gain by finding the entropy for tuple.
o Now find the best split value of that particular attribute and save it in the TreeNode.
o Now as per the split separate the dataset into two parts - Left and Right and then recursively find the attribute with maximum gain with repeating the steps above.
o At the end of the recursion we will have the model built with leaf nodes representing the class variables - 'B', 'G', 'M', 'N'
• After the tree is built, we save the tree using pickle package in python
• To test the model on test dataset, we find confusion matrix and the accuracy of the model as below-
o Key concept is that we run our model using the classify() method which will traverse the model as per the each instance of the test dataset and finds out the predicted class. o Accuracy is calculated using the number of instances

1.1 Code Image:

```
File: kNN.py
This file contains a single class definition from which
k-nearest neighbor models can be instantiated.
This class implements two methods: train and predict.
While the k-nearest neighbor algorithm does not require training,
the api has been defined like this to remain consistent with other
open-source machine learning libraries like scikit-learn.
"""


import numpy as np
from src.distance import euclidean


class KNearestNeighbors:

    def __init__(self, k=3, distance_metric=euclidean):
        """Initialize k value and distance metric used for model."""
        self.k = k
        self.distance = distance_metric
        self.data = None

    def train(self, X, y):
        """Zip labels and input data together for classification."""
        # raise value error if inputs are wrong length or different types
        if len(X) != len(y) or type(X) != type(y):
            raise ValueError("X and y are incompatible.")
        # convert ndarrays to lists
        if type(X) == np.ndarray:
            X, y = X.tolist(), y.tolist()
        # set data attribute containing instances and labels
        self.data = [X[i]+[y[i]] for i in range(len(X))]

    def predict(self, a):
        """Predict class based on k-nearest neighbors."""
        neighbors = []
        # create mapping from distance to instance
        distances = {self.distance(x[:-1], a): x for x in self.data}
        # collect classes of k instances with shortest distance
        for key in sorted(distances.keys())[:self.k]:
            neighbors.append(distances[key][-1])
        # return most common vote
        return max(set(neighbors), key=neighbors.count)
```

2 Decision Tree:

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements

Algorithm Explanation

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too. The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data. In Decision Trees, for predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with the records attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

2.1 Code Screenshot

```
9       def __init__(self,split,col_index):
10          self.col_id= col_index
11          self.split_value= split
12          self.parent=None
13          self.left= None
14          self.right= None
15
16  class Tree():
17
18      def __init__(self):
19          self.treemodel = None
20
21      def train(self,trainData):
22          #Attributes/Last Column is class
23          self.createTree(trainData)
24
25
26      def createTree(self,trainData):
27          #create the tree
28          self.treemodel=build_tree(trainData,[])
29          saveTree(self.treemodel)
30
31      def accuracy_confusion_matrix(self,testData):
32          #prints the tree confusion matrix along with the accuracy
33          build_confusion_matrix(self.treemodel,testData)
34
35  #returns the best split on the data instance along
36  #with the splitted dataset and column index
37  def getBestSplit(data):
38      #set the max information gain
39      maxInfoGain = -float('inf')
40
41      #convert to array
42      dataArray = np.asarray(data)
43
44      #to extract rows and columns
45      dimension = np.shape(dataArray)
46
47      #iterate through the matrix
48      for col in range(dimension[1]-1):
49         dataArray = sorted(dataArray, key=lambda x: x[col])
50         for row in range(dimension[0]-1):
51             val1=dataArray[row][col]
52             val2=dataArray[row+1][col]
53             expectedSplit = (float(val1)+float(val2))/2.0
54             infoGain,l,r= calcInfoGain(data,col,expectedSplit)
55             if(infoGain>maxInfoGain):
56                  maxInfoGain=infoGain
57                  best= (col,expectedSplit,l,r)
58      return best
59
60  #This method is used to calculate the gain and returns
61  #the left and right data as per the split
62  def calcInfoGain(data,col,split):
63      totallen = len(data)
64      infoGain = entropy(data)
65
66      left_data, right_data =getDataSplit(data,split,col)
```

3 Conclusion

I would like to thank to my teacher for giving me this amazing opportunity to learn these algorithms. I am sure this would be of help for my career of programming.