|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | |
|  | Name of the student: | | Yash Sankpal | | | | Roll No. | | 8695 | |  |
| Practical Number: | | 7 | | | | Date of Practical: | |  | |
| Relevant CO’s | | At the end of the course students will be able to apply appropriate algorithms for extracting knowledge from given dataset. | | | | | | | |
| Sign here to indicate that you have read all the relevant material provided Sign:  before attempting this practical  Practical grading using Rubrics | | | | | | | | | |
| Indicator | Very Poor | | Poor | Average | Good | | Excellent | |  |
| Timeline  (2) | Practical not submitted (0) | | More than  two session  late (0.5) | Two sessions late (1) | One session  late (1.5) | | Early or on  time (2) | |
| Code de-  sign (3) | N/A | | Very poor code de-  sign(0) | poor design  (1) | design with good coding standards (2) | | Accurate Design  with better coding  standards(3) | |
| Execution  (3) | N/A | | Very less execution (0) | little execu-  tion.(1) | Major execu-  tion(2) | | Entire code execution (3) | |
| Postlab (2) | Both an-  swers wrong(0) | | N/A | One answer  correct (1) | N/A | | Both an-  swers correct (2) | |
| |  |  | | --- | --- | | Total Marks (10) | Sign of instructor with date | |  |  | | | | | | | | | |

|  |
| --- |
| Practical  Course title: Big Data Analytics  Course term: 2021-2022  Problem Statement: To implement kNN algorithm using map-reduce.  Theory:  The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.  In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point.  A commonly used distance metric for continuous variables is Euclidean distance. For discrete variables, such as for text classification, another metric can be used, such as the overlap metric (or Hamming distance). In the context of gene expression microarray data, for example, k-NN has been employed with correlation coefficients, such as Pearson and Spearman, as a metric.[6] Often, the classification accuracy of k-NN can be improved significantly if the distance metric is learned with specialized algorithms such as Large Margin Nearest Neighbor or Neighbourhood components analysis.  A drawback of the basic "majority voting" classification occurs when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the k nearest neighbors due to their large number.[7] One way to overcome this problem is to weight the classification, taking into account the distance from the test point to each of its k nearest neighbors. The class (or value, in regression problems) of each of the k nearest points is multiplied by a weight proportional to the inverse of the distance from that point to the test point. Another way to overcome skew is by abstraction in data representation. For example, in a self-organizing map (SOM), each node is a representative (a center) of a cluster of similar points, regardless of their density in the original training data. K-NN can then be applied to the SOM. |

|  |
| --- |
| Code  Write map-reduce code to implement kNN algorithm. Use CarOwners.csv as input file to train the model.  Take the following tuple as input from the user and try classifying it using kNN.  Test tuple to be taken as input from the user: k=5, Age=67, income=16668, Status=Single, Gender=Male,  Children=3.  code for mapper: |
| public class Mapper7 extends Mapper<Object, Text, NullWritable, Model> {  Model model = new Model();  TreeMap<Double, String> KnnMap = new TreeMap<Double, String>();  public void map(Object ikey, Text ivalue, Context context) throws IOException, InterruptedException {    String value[] = ivalue.toString().split(",");    Double tDist = Data.totalSquaredDistance(value[0],value[1],value[2],value[3],value[4]);    KnnMap.put(tDist,value[5]);    if(KnnMap.size() > Data.K) {  KnnMap.remove(KnnMap.lastKey());  }  }  @Override  protected void cleanup(Context context) throws IOException,InterruptedException{  for(Map.Entry<Double, String> entry : KnnMap.entrySet()) {  Double knnDist = entry.getKey();  String knnModel = entry.getValue();    model.set(knnDist,knnModel);    context.write(NullWritable.get(), model);  }  }  } |
| Code for Reducer: |
| public class Reducer7 extends Reducer<NullWritable, Model, NullWritable, Text> {  TreeMap<Double,String> KnnMap = new TreeMap<Double,String>();  public void reduce(NullWritable \_key, Iterable<Model> values, Context context) throws IOException, InterruptedException {  // process values  for (Model val : values)  {  String rModel = val.getModel();  double tDist = val.getDistance();    // Populate another TreeMap with the distance and model information extracted from the  // DoubleString objects and trim it to size K as before.  KnnMap.put(tDist, rModel);  if (KnnMap.size() > Data.*K*)  {  KnnMap.remove(KnnMap.lastKey());  }  }  // This section determines which of the K values (models) in the TreeMap occurs most frequently  // by means of constructing an intermediate ArrayList and HashMap.  // A List of all the values in the TreeMap.  List<String> knnList = new ArrayList<String>(KnnMap.values());  Map<String, Integer> freqMap = new HashMap<String, Integer>();    // Add the members of the list to the HashMap as keys and the number of times each occurs  // (frequency) as values  for(int i=0; i< knnList.size(); i++)  {  Integer frequency = freqMap.get(knnList.get(i));  if(frequency == null)  {  freqMap.put(knnList.get(i), 1);  } else  {  freqMap.put(knnList.get(i), frequency+1);  }  }    // Examine the HashMap to determine which key (model) has the highest value (frequency)  String mostCommonModel = null;  int maxFrequency = -1;  for(Map.Entry<String, Integer> entry: freqMap.entrySet())  {  if(entry.getValue() > maxFrequency)  {  mostCommonModel = entry.getKey();  maxFrequency = entry.getValue();  }  }    // Finally write to context another NullWritable as key and the most common model just counted as value.  context.write(NullWritable.*get*(), new Text(mostCommonModel)); // Use this line to produce a single classification  }  } |
| Code for Driver Class: |
| public class Driver7 {  public static void main(String[] args) throws Exception {    Configuration conf = new Configuration();    Path output = new Path(Data.***outputFile***);  FileSystem fs = FileSystem.*get*(output.toUri(),conf);    if(fs.exists(output)) {  fs.delete(output,true);  }  // Data.readTestData();    Job job = Job.*getInstance*(conf, "K-Nearest Neighbour");    job.setJarByClass(Driver7.class);  // **TODO**: specify a mapper  job.setMapperClass(Mapper7.class);  // **TODO**: specify a reducer  job.setReducerClass(Reducer7.class);  job.setNumReduceTasks(1);  // **TODO**: specify output types  job.setMapOutputKeyClass(NullWritable.class);  job.setMapOutputValueClass(Model.class);  job.setOutputKeyClass(NullWritable.class);  job.setOutputValueClass(Text.class);  // **TODO**: specify input and output DIRECTORIES (not files)  FileInputFormat.*setInputPaths*(job, new Path(Data.***inputFile***));  FileOutputFormat.*setOutputPath*(job, new Path(Data.***outputFile***));  System.***out***.println(job.waitForCompletion(true));  }  } |
|  |

PostLab:

Try using k=3 in above code. Comment on the observation made.

Answer for postlab question

When k=3

The test data was put under MX5 car model category.

Try using k=7 in above code. Comment on the observation made.

Answer for postlab question

The test data was put under BMW5 car model category.