**Programming Project: k-Nearest Neighbor Method**

package com.javferna.packtpub.mastering.knn.main;  
  
import java.util.Date;  
import java.util.List;  
  
import com.javferna.packtpub.mastering.knn.data.BankMarketing;  
import com.javferna.packtpub.mastering.knn.loader.BankMarketingLoader;  
import com.javferna.packtpub.mastering.knn.serial.KnnClassifier;  
  
*/\*\*  
 \* Main class that launches the tests using the serial knn with serial sorting  
 \** ***@author*** *Usuario  
 \*  
 \*/*public class SerialMain {  
  
 public static void main(String[] args) {  
  
 BankMarketingLoader loader = new BankMarketingLoader();  
 List<BankMarketing> train = loader.load("data\\bank.data");  
 System.*out*.println("Train: " + train.size());  
 List<BankMarketing> test = loader.load("data\\bank.test");  
 System.*out*.println("Test: " + test.size());  
 double currentTime = 0d;  
 int success = 0, mistakes = 0;  
   
 int k = 10;  
 if (args.length==1) {  
 k = Integer.*parseInt*(args[0]);  
 }  
  
 success = 0;  
 mistakes = 0;  
 KnnClassifier classifier = new KnnClassifier(train, k);  
 try {  
 Date start, end;  
 start = new Date();  
 for (BankMarketing example : test) {  
 String tag = classifier.classify(example);  
 if (tag.equals(example.getTag())) {  
 success++;  
 } else {  
 mistakes++;  
 }  
 }  
 end = new Date();  
  
 currentTime = end.getTime() - start.getTime();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.*out*.println("Serial Classifier - K: " + k);  
 System.*out*.println("Success: " + success);  
 System.*out*.println("Mistakes: " + mistakes);  
 System.*out*.println("Execution Time: " + (currentTime / 1000)  
 + " seconds.");  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
  
 }  
  
}

package com.javferna.packtpub.mastering.knn.serial;  
  
import java.util.Arrays;  
import java.util.Collections;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
import com.javferna.packtpub.mastering.knn.data.Distance;  
import com.javferna.packtpub.mastering.knn.data.Sample;  
import com.javferna.packtpub.mastering.knn.distances.EuclideanDistanceCalculator;  
  
*/\*\*  
 \* Serial implementation of the Knn algorithm  
 \** ***@author*** *author  
 \*/*public class KnnClassifier {  
  
 */\*\*  
 \* List of train data  
 \*/* private final List<? extends Sample> dataSet;  
   
 */\*\*  
 \* K parameter  
 \*/* private int k;  
   
 */\*\*  
 \* Constructor of the class. Initialize the internal data  
 \** ***@param*** *dataSet Train data  
 \** ***@param*** *k K parameter  
 \*/* public KnnClassifier(List<? extends Sample> dataSet, int k) {  
 this.dataSet=dataSet;  
 this.k=k;  
 }  
   
 */\*\*  
 \* Method that classifies an example  
 \** ***@param*** *example Example to classify  
 \** ***@return*** *The tag or class of the example  
 \** ***@throws*** *Exception Exception if something goes wrong  
 \*/* public String classify (Sample example) {  
   
 Distance[] distances=new Distance[dataSet.size()];  
   
 int index=0;  
   
 for (Sample localExample : dataSet) {  
 distances[index]=new Distance();  
 distances[index].setIndex(index);  
 distances[index].setDistance(EuclideanDistanceCalculator.*calculate*(localExample, example));  
 index++;  
 }  
  
 Arrays.*sort*(distances);  
   
 Map<String, Integer> results = new HashMap<>();  
 for (int i = 0; i < k; i++) {  
 Sample localExample = dataSet.get(distances[i].getIndex());  
 String tag = localExample.getTag();  
 results.merge(tag, 1, (a, b) -> a+b);  
 }  
  
   
   
 return Collections.*max*(results.entrySet(),  
 Map.Entry.*comparingByValue*()).getKey();  
 }  
}

package com.javferna.packtpub.mastering.knn.data;  
  
*/\*\*  
 \* This class implements the example of the Bank Marketing problem with all the attributes.  
 \** ***@author*** *author  
 \*  
 \*/*public class BankMarketing extends Sample{  
  
 private byte age;  
 private byte jobAdmin;  
 private byte jobBlueCollar;  
 private byte jobEntrepreneur;  
 private byte jobHousemaid;  
 private byte jobManagement;  
 private byte jobRetired;  
 private byte jobSelfEmployed;  
 private byte jobServices;  
 private byte jobStudent;  
 private byte jobTechnician;  
 private byte jobUnemployed;  
 private byte jobUnknown;  
 private byte maritalDivorced;  
 private byte maritalMarried;  
 private byte maritalSingle;  
 private byte maritalUnknown;  
 private byte educationBasic4y;  
 private byte educationBasic6y;  
 private byte educationBasic9y;  
 private byte educationHighSchool;  
 private byte educationIlliterate;  
 private byte educationProfessionalCourse;  
 private byte educationUniversityDegree;  
 private byte educationUnknown;  
 private byte creditNo;  
 private byte creditYes;  
 private byte creditUnknown;  
 private byte housingNo;  
 private byte housingYes;  
 private byte housingUnknown;  
 private byte loanNo;  
 private byte loanYes;  
 private byte loanUnknown;  
 private byte contactCellular;  
 private byte contactTelephone;  
 private byte contactJan;  
 private byte contactFeb;  
 private byte contactMar;  
 private byte contactApr;  
 private byte contactMay;  
 private byte contactJun;  
 private byte contactJul;  
 private byte contactAug;  
 private byte contactSep;  
 private byte contactOct;  
 private byte contactNov;  
 private byte contactDec;  
 private byte contactMon;  
 private byte contactTue;  
 private byte contactWed;  
 private byte contactThu;  
 private byte contactFri;  
 private int duration;  
 private byte campaign;  
 private int pdays;  
 private byte pdaysNever;  
 private byte previous;  
 private byte poutcomeFailure;  
 private byte poutcomeNonexistent;  
 private byte poutcomeSuccess;  
 private float empVarRate;  
 private float consPriceIdx;  
 private float consConfIdx;  
 private float euribor3m;  
 private float nrEmployed;  
 private String target;  
   
   
 */\*\*  
 \* Method that stablish the values of the attributes from an array of Strings  
 \** ***@param*** *data Array of Strings with the values of the attributes  
 \** ***@throws*** *Exception Exception if something goes wrong  
 \*/* public void setData(String []data) throws Exception {  
   
 if (data.length!=67) {  
 throw new Exception("Wrong data length: "+data.length);  
 }  
 age=Byte.*valueOf*(data[0]);  
 jobAdmin=Byte.*valueOf*(data[1]);  
 jobBlueCollar=Byte.*valueOf*(data[2]);  
 jobEntrepreneur=Byte.*valueOf*(data[3]);  
 jobHousemaid=Byte.*valueOf*(data[4]);  
 jobManagement=Byte.*valueOf*(data[5]);  
 jobRetired=Byte.*valueOf*(data[6]);  
 jobSelfEmployed=Byte.*valueOf*(data[7]);  
 jobServices=Byte.*valueOf*(data[8]);  
 jobStudent=Byte.*valueOf*(data[9]);  
 jobTechnician=Byte.*valueOf*(data[10]);  
 jobUnemployed=Byte.*valueOf*(data[11]);  
 jobUnknown=Byte.*valueOf*(data[12]);  
 maritalDivorced=Byte.*valueOf*(data[13]);  
 maritalMarried=Byte.*valueOf*(data[14]);  
 maritalSingle=Byte.*valueOf*(data[15]);  
 maritalUnknown=Byte.*valueOf*(data[16]);  
 educationBasic4y=Byte.*valueOf*(data[17]);  
 educationBasic6y=Byte.*valueOf*(data[18]);  
 educationBasic9y=Byte.*valueOf*(data[19]);  
 educationHighSchool=Byte.*valueOf*(data[20]);  
 educationIlliterate=Byte.*valueOf*(data[21]);  
 educationProfessionalCourse=Byte.*valueOf*(data[22]);  
 educationUniversityDegree=Byte.*valueOf*(data[23]);  
 educationUnknown=Byte.*valueOf*(data[24]);  
 creditNo=Byte.*valueOf*(data[25]);  
 creditYes=Byte.*valueOf*(data[26]);  
 creditUnknown=Byte.*valueOf*(data[27]);  
 housingNo=Byte.*valueOf*(data[28]);  
 housingYes=Byte.*valueOf*(data[29]);  
 housingUnknown=Byte.*valueOf*(data[30]);  
 loanNo=Byte.*valueOf*(data[31]);  
 loanYes=Byte.*valueOf*(data[32]);  
 loanUnknown=Byte.*valueOf*(data[33]);  
 contactCellular=Byte.*valueOf*(data[34]);  
 contactTelephone=Byte.*valueOf*(data[35]);  
 contactJan=Byte.*valueOf*(data[36]);  
 contactFeb=Byte.*valueOf*(data[37]);  
 contactMar=Byte.*valueOf*(data[38]);  
 contactApr=Byte.*valueOf*(data[39]);  
 contactMay=Byte.*valueOf*(data[40]);  
 contactJun=Byte.*valueOf*(data[41]);  
 contactJul=Byte.*valueOf*(data[42]);  
 contactAug=Byte.*valueOf*(data[43]);  
 contactSep=Byte.*valueOf*(data[44]);  
 contactOct=Byte.*valueOf*(data[45]);  
 contactNov=Byte.*valueOf*(data[46]);  
 contactDec=Byte.*valueOf*(data[47]);  
 contactMon=Byte.*valueOf*(data[48]);  
 contactTue=Byte.*valueOf*(data[49]);  
 contactWed=Byte.*valueOf*(data[50]);  
 contactThu=Byte.*valueOf*(data[51]);  
 contactFri=Byte.*valueOf*(data[52]);  
 duration=Integer.*valueOf*(data[53]);  
 campaign=Byte.*valueOf*(data[54]);  
 pdays=Integer.*valueOf*(data[55]);  
 pdaysNever=Byte.*valueOf*(data[56]);  
 previous=Byte.*valueOf*(data[57]);  
 poutcomeFailure=Byte.*valueOf*(data[58]);  
 poutcomeNonexistent=Byte.*valueOf*(data[59]);  
 poutcomeSuccess=Byte.*valueOf*(data[60]);  
 empVarRate=Float.*valueOf*(data[61]);  
 consPriceIdx=Float.*valueOf*(data[62]);  
 consConfIdx=Float.*valueOf*(data[63]);  
 euribor3m=Float.*valueOf*(data[64]);  
 nrEmployed=Float.*valueOf*(data[65]);  
 target=data[66];  
   
 }  
  
 public byte getAge() {  
 return age;  
 }  
  
 public void setAge(byte age) {  
 this.age = age;  
 }  
  
 public byte getJobAdmin() {  
 return jobAdmin;  
 }  
  
 public void setJobAdmin(byte jobAdmin) {  
 this.jobAdmin = jobAdmin;  
 }  
  
 public byte getJobBlueCollar() {  
 return jobBlueCollar;  
 }  
  
 public void setJobBlueCollar(byte jobBlueCollar) {  
 this.jobBlueCollar = jobBlueCollar;  
 }  
  
 public byte getJobEntrepreneur() {  
 return jobEntrepreneur;  
 }  
  
 public void setJobEntrepreneur(byte jobEntrepreneur) {  
 this.jobEntrepreneur = jobEntrepreneur;  
 }  
  
 public byte getJobHousemaid() {  
 return jobHousemaid;  
 }  
  
 public void setJobHousemaid(byte jobHousemaid) {  
 this.jobHousemaid = jobHousemaid;  
 }  
  
 public byte getJobManagement() {  
 return jobManagement;  
 }  
  
 public void setJobManagement(byte jobManagement) {  
 this.jobManagement = jobManagement;  
 }  
  
 public byte getJobRetired() {  
 return jobRetired;  
 }  
  
 public void setJobRetired(byte jobRetired) {  
 this.jobRetired = jobRetired;  
 }  
  
 public byte getJobSelfEmployed() {  
 return jobSelfEmployed;  
 }  
  
 public void setJobSelfEmployed(byte jobSelfEmployed) {  
 this.jobSelfEmployed = jobSelfEmployed;  
 }  
  
 public byte getJobServices() {  
 return jobServices;  
 }  
  
 public void setJobServices(byte jobServices) {  
 this.jobServices = jobServices;  
 }  
  
 public byte getJobStudent() {  
 return jobStudent;  
 }  
  
 public void setJobStudent(byte jobStudent) {  
 this.jobStudent = jobStudent;  
 }  
  
 public byte getJobTechnician() {  
 return jobTechnician;  
 }  
  
 public void setJobTechnician(byte jobTechnician) {  
 this.jobTechnician = jobTechnician;  
 }  
  
 public byte getJobUnemployed() {  
 return jobUnemployed;  
 }  
  
 public void setJobUnemployed(byte jobUnemployed) {  
 this.jobUnemployed = jobUnemployed;  
 }  
  
 public byte getJobUnknown() {  
 return jobUnknown;  
 }  
  
 public void setJobUnknown(byte jobUnknown) {  
 this.jobUnknown = jobUnknown;  
 }  
  
 public byte getMaritalDivorced() {  
 return maritalDivorced;  
 }  
  
 public void setMaritalDivorced(byte maritalDivorced) {  
 this.maritalDivorced = maritalDivorced;  
 }  
  
 public byte getMaritalMarried() {  
 return maritalMarried;  
 }  
  
 public void setMaritalMarried(byte maritalMarried) {  
 this.maritalMarried = maritalMarried;  
 }  
  
 public byte getMaritalSingle() {  
 return maritalSingle;  
 }  
  
 public void setMaritalSingle(byte maritalSingle) {  
 this.maritalSingle = maritalSingle;  
 }  
  
 public byte getMaritalUnknown() {  
 return maritalUnknown;  
 }  
  
 public void setMaritalUnknown(byte maritalUnknown) {  
 this.maritalUnknown = maritalUnknown;  
 }  
  
 public byte getEducationBasic4y() {  
 return educationBasic4y;  
 }  
  
 public void setEducationBasic4y(byte educationBasic4y) {  
 this.educationBasic4y = educationBasic4y;  
 }  
  
 public byte getEducationBasic6y() {  
 return educationBasic6y;  
 }  
  
 public void setEducationBasic6y(byte educationBasic6y) {  
 this.educationBasic6y = educationBasic6y;  
 }  
  
 public byte getEducationBasic9y() {  
 return educationBasic9y;  
 }  
  
 public void setEducationBasic9y(byte educationBasic9y) {  
 this.educationBasic9y = educationBasic9y;  
 }  
  
 public byte getEducationHighSchool() {  
 return educationHighSchool;  
 }  
  
 public void setEducationHighSchool(byte educationHighSchool) {  
 this.educationHighSchool = educationHighSchool;  
 }  
  
 public byte getEducationIlliterate() {  
 return educationIlliterate;  
 }  
  
 public void setEducationIlliterate(byte educationIlliterate) {  
 this.educationIlliterate = educationIlliterate;  
 }  
  
 public byte getEducationProfessionalCourse() {  
 return educationProfessionalCourse;  
 }  
  
 public void setEducationProfessionalCourse(byte educationProfessionalCourse) {  
 this.educationProfessionalCourse = educationProfessionalCourse;  
 }  
  
 public byte getEducationUniversityDegree() {  
 return educationUniversityDegree;  
 }  
  
 public void setEducationUniversityDegree(byte educationUniversityDegree) {  
 this.educationUniversityDegree = educationUniversityDegree;  
 }  
  
 public byte getEducationUnknown() {  
 return educationUnknown;  
 }  
  
 public void setEducationUnknown(byte educationUnknown) {  
 this.educationUnknown = educationUnknown;  
 }  
  
 public byte getCreditNo() {  
 return creditNo;  
 }  
  
 public void setCreditNo(byte creditNo) {  
 this.creditNo = creditNo;  
 }  
  
 public byte getCreditYes() {  
 return creditYes;  
 }  
  
 public void setCreditYes(byte creditYes) {  
 this.creditYes = creditYes;  
 }  
  
 public byte getCreditUnknown() {  
 return creditUnknown;  
 }  
  
 public void setCreditUnknown(byte creditUnknown) {  
 this.creditUnknown = creditUnknown;  
 }  
  
 public byte getHousingNo() {  
 return housingNo;  
 }  
  
 public void setHousingNo(byte housingNo) {  
 this.housingNo = housingNo;  
 }  
  
 public byte getHousingYes() {  
 return housingYes;  
 }  
  
 public void setHousingYes(byte housingYes) {  
 this.housingYes = housingYes;  
 }  
  
 public byte getHousingUnknown() {  
 return housingUnknown;  
 }  
  
 public void setHousingUnknown(byte housingUnknown) {  
 this.housingUnknown = housingUnknown;  
 }  
  
 public byte getLoanNo() {  
 return loanNo;  
 }  
  
 public void setLoanNo(byte loanNo) {  
 this.loanNo = loanNo;  
 }  
  
 public byte getLoanYes() {  
 return loanYes;  
 }  
  
 public void setLoanYes(byte loanYes) {  
 this.loanYes = loanYes;  
 }  
  
 public byte getLoanUnknown() {  
 return loanUnknown;  
 }  
  
 public void setLoanUnknown(byte loanUnknown) {  
 this.loanUnknown = loanUnknown;  
 }  
  
 public byte getContactCellular() {  
 return contactCellular;  
 }  
  
 public void setContactCellular(byte contactCellular) {  
 this.contactCellular = contactCellular;  
 }  
  
 public byte getContactTelephone() {  
 return contactTelephone;  
 }  
  
 public void setContactTelephone(byte contactTelephone) {  
 this.contactTelephone = contactTelephone;  
 }  
  
 public byte getContactJan() {  
 return contactJan;  
 }  
  
 public void setContactJan(byte contactJan) {  
 this.contactJan = contactJan;  
 }  
  
 public byte getContactFeb() {  
 return contactFeb;  
 }  
  
 public void setContactFeb(byte contactFeb) {  
 this.contactFeb = contactFeb;  
 }  
  
 public byte getContactMar() {  
 return contactMar;  
 }  
  
 public void setContactMar(byte contactMar) {  
 this.contactMar = contactMar;  
 }  
  
 public byte getContactApr() {  
 return contactApr;  
 }  
  
 public void setContactApr(byte contactApr) {  
 this.contactApr = contactApr;  
 }  
  
 public byte getContactMay() {  
 return contactMay;  
 }  
  
 public void setContactMay(byte contactMay) {  
 this.contactMay = contactMay;  
 }  
  
 public byte getContactJun() {  
 return contactJun;  
 }  
  
 public void setContactJun(byte contactJun) {  
 this.contactJun = contactJun;  
 }  
  
 public byte getContactJul() {  
 return contactJul;  
 }  
  
 public void setContactJul(byte contactJul) {  
 this.contactJul = contactJul;  
 }  
  
 public byte getContactAug() {  
 return contactAug;  
 }  
  
 public void setContactAug(byte contactAug) {  
 this.contactAug = contactAug;  
 }  
  
 public byte getContactSep() {  
 return contactSep;  
 }  
  
 public void setContactSep(byte contactSep) {  
 this.contactSep = contactSep;  
 }  
  
 public byte getContactOct() {  
 return contactOct;  
 }  
  
 public void setContactOct(byte contactOct) {  
 this.contactOct = contactOct;  
 }  
  
 public byte getContactNov() {  
 return contactNov;  
 }  
  
 public void setContactNov(byte contactNov) {  
 this.contactNov = contactNov;  
 }  
  
 public byte getContactDec() {  
 return contactDec;  
 }  
  
 public void setContactDec(byte contactDec) {  
 this.contactDec = contactDec;  
 }  
  
 public byte getContactMon() {  
 return contactMon;  
 }  
  
 public void setContactMon(byte contactMon) {  
 this.contactMon = contactMon;  
 }  
  
 public byte getContactTue() {  
 return contactTue;  
 }  
  
 public void setContactTue(byte contactTue) {  
 this.contactTue = contactTue;  
 }  
  
 public byte getContactWed() {  
 return contactWed;  
 }  
  
 public void setContactWed(byte contactWed) {  
 this.contactWed = contactWed;  
 }  
  
 public byte getContactThu() {  
 return contactThu;  
 }  
  
 public void setContactThu(byte contactThu) {  
 this.contactThu = contactThu;  
 }  
  
 public byte getContactFri() {  
 return contactFri;  
 }  
  
 public void setContactFri(byte contactFri) {  
 this.contactFri = contactFri;  
 }  
  
 public int getDuration() {  
 return duration;  
 }  
  
 public void setDuration(int duration) {  
 this.duration = duration;  
 }  
  
 public byte getCampaign() {  
 return campaign;  
 }  
  
 public void setCampaign(byte campaign) {  
 this.campaign = campaign;  
 }  
  
 public int getPdays() {  
 return pdays;  
 }  
  
 public void setPdays(int pdays) {  
 this.pdays = pdays;  
 }  
  
 public byte getPdaysNever() {  
 return pdaysNever;  
 }  
  
 public void setPdaysNever(byte pdaysNever) {  
 this.pdaysNever = pdaysNever;  
 }  
  
 public byte getPrevious() {  
 return previous;  
 }  
  
 public void setPrevious(byte previous) {  
 this.previous = previous;  
 }  
  
 public byte getPoutcomeFailure() {  
 return poutcomeFailure;  
 }  
  
 public void setPoutcomeFailure(byte poutcomeFailure) {  
 this.poutcomeFailure = poutcomeFailure;  
 }  
  
 public byte getPoutcomeNonexistent() {  
 return poutcomeNonexistent;  
 }  
  
 public void setPoutcomeNonexistent(byte poutcomeNonexistent) {  
 this.poutcomeNonexistent = poutcomeNonexistent;  
 }  
  
 public byte getPoutcomeSuccess() {  
 return poutcomeSuccess;  
 }  
  
 public void setPoutcomeSuccess(byte poutcomeSuccess) {  
 this.poutcomeSuccess = poutcomeSuccess;  
 }  
  
 public float getEmpVarRate() {  
 return empVarRate;  
 }  
  
 public void setEmpVarRate(float empVarRate) {  
 this.empVarRate = empVarRate;  
 }  
  
 public float getConsPriceIdx() {  
 return consPriceIdx;  
 }  
  
 public void setConsPriceIdx(float consPriceIdx) {  
 this.consPriceIdx = consPriceIdx;  
 }  
  
 public float getConsConfIdx() {  
 return consConfIdx;  
 }  
  
 public void setConsConfIdx(float consConfIdx) {  
 this.consConfIdx = consConfIdx;  
 }  
  
 public float getEuribor3m() {  
 return euribor3m;  
 }  
  
 public void setEuribor3m(float euribor3m) {  
 this.euribor3m = euribor3m;  
 }  
  
 public float getNrEmployed() {  
 return nrEmployed;  
 }  
  
 public void setNrEmployed(float nrEmployed) {  
 this.nrEmployed = nrEmployed;  
 }  
  
 public String getTarget() {  
 return target;  
 }  
  
 public void setTarget(String target) {  
 this.target = target;  
 }  
  
 @Override  
 public String getTag() {  
 return target;  
 }  
  
 @Override  
 public double[] getExample() {  
 double ret[]=new double[66];  
 ret[0]=age;  
 ret[1]=jobAdmin;  
 ret[2]=jobBlueCollar;  
 ret[3]=jobEntrepreneur;  
 ret[4]=jobHousemaid;  
 ret[5]=jobManagement;  
 ret[6]=jobRetired;  
 ret[7]=jobSelfEmployed;  
 ret[8]=jobServices;  
 ret[9]=jobStudent;  
 ret[10]=jobTechnician;  
 ret[11]=jobUnemployed;  
 ret[12]=jobUnknown;  
 ret[13]=maritalDivorced;  
 ret[14]=maritalMarried;  
 ret[15]=maritalSingle;  
 ret[16]=maritalUnknown;  
 ret[17]=educationBasic4y;  
 ret[18]=educationBasic6y;  
 ret[19]=educationBasic9y;  
 ret[20]=educationHighSchool;  
 ret[21]=educationIlliterate;  
 ret[22]=educationProfessionalCourse;  
 ret[23]=educationUniversityDegree;  
 ret[24]=educationUnknown;  
 ret[25]=creditNo;  
 ret[26]=creditYes;  
 ret[27]=creditUnknown;  
 ret[28]=housingNo;  
 ret[29]=housingYes;  
 ret[30]=housingUnknown;  
 ret[31]=loanNo;  
 ret[32]=loanYes;  
 ret[33]=loanUnknown;  
 ret[34]=contactCellular;  
 ret[35]=contactTelephone;  
 ret[36]=contactJan;  
 ret[37]=contactFeb;  
 ret[38]=contactMar;  
 ret[39]=contactApr;  
 ret[40]=contactMay;  
 ret[41]=contactJun;  
 ret[42]=contactJul;  
 ret[43]=contactAug;  
 ret[44]=contactSep;  
 ret[45]=contactOct;  
 ret[46]=contactNov;  
 ret[47]=contactDec;  
 ret[48]=contactMon;  
 ret[49]=contactTue;  
 ret[50]=contactWed;  
 ret[51]=contactThu;  
 ret[52]=contactFri;  
 ret[53]=duration;  
 ret[54]=campaign;  
 ret[55]=pdays;  
 ret[56]=pdaysNever;  
 ret[57]=previous;  
 ret[58]=poutcomeFailure;  
 ret[59]=poutcomeNonexistent;  
 ret[60]=poutcomeSuccess;  
 ret[61]=empVarRate;  
 ret[62]=consPriceIdx;  
 ret[63]=consConfIdx;  
 ret[64]=euribor3m;  
 ret[65]=nrEmployed;  
 return ret;  
 }  
   
   
   
}

package com.javferna.packtpub.mastering.knn.data;  
  
*/\*\*  
 \* This class stores the distance between an example of the train data  
 \* and the example the algorithm wants to classify  
 \** ***@author*** *author  
 \*  
 \*/*public class Distance implements Comparable<Distance> {  
  
 */\*\*  
 \* Index of the train instance  
 \*/* private int index;  
 */\*\*  
 \* Distance between the train instance and the example  
 \*/* private double distance;  
   
   
 */\*\*  
 \* Method that compare two Distance objects  
 \** ***@param*** *other: The other object to calculate the distance  
 \** ***@return*** *The distance between this instance and the other instance  
 \*/* @Override  
 public int compareTo(Distance other) {  
 if (this.distance < other.getDistance()) {  
 return -1;  
 } else if (this.distance > other.getDistance()) {  
 return 1;  
 }  
 return 0;  
 }  
  
 */\*\*  
 \* Method that returns the index  
 \** ***@return*** *The index  
 \*/* public int getIndex() {  
 return index;  
 }  
  
 */\*\*  
 \* Method that sets the index  
 \** ***@param*** *index The index  
 \*/* public void setIndex(int index) {  
 this.index = index;  
 }  
  
 */\*\*  
 \* Method that returns the distance  
 \** ***@return*** *The distance  
 \*/* public double getDistance() {  
 return distance;  
 }  
  
 */\*\*  
 \* Method that sets the distance  
 \** ***@param*** *distance The distance  
 \*/* public void setDistance(double distance) {  
 this.distance = distance;  
 }  
  
   
   
}

package com.javferna.packtpub.mastering.knn.data;  
  
*/\*\*  
 \* Abstract class that defines the basic elements of an example to the knn algoritm  
 \** ***@author*** *author  
 \*  
 \*/*public abstract class Sample {  
  
 */\*\*  
 \* Method that returns the tag or class of the example  
 \** ***@return*** *The tag or class of the examples  
 \*/* public abstract String getTag();  
   
 */\*\*  
 \* Method that return the values of the attributes of the example as an array of doubles  
 \** ***@return*** *The values of the attributes of the example  
 \*/* public abstract double[] getExample();  
}

package com.javferna.packtpub.mastering.knn.distances;  
  
import com.javferna.packtpub.mastering.knn.data.Sample;  
  
*/\*\*  
 \* This class calculates the distance between two examples using the euclidean formula  
 \** ***@author*** *author  
 \*  
 \*/*public class EuclideanDistanceCalculator {  
   
 */\*\*  
 \* Method that calculates the euclidean distance between two examples  
 \** ***@param*** *example1 One of the examples  
 \** ***@param*** *example2 The other example  
 \** ***@return*** *The euclidean distance between the two examples  
 \** ***@throws*** *Exception Exception if something goes wrong  
 \*/* public static double calculate (Sample example1, Sample example2) {  
 double ret=0.0d;  
   
 double[] data1=example1.getExample();  
 double[] data2=example2.getExample();  
   
 if (data1.length!=data2.length) {  
 throw new IllegalArgumentException ("Vector doesn't have the same length");  
 }  
   
 for (int i=0; i<data1.length; i++) {  
 ret+=Math.*pow*(data1[i]-data2[i], 2);  
 }  
 return Math.*sqrt*(ret);  
  
 }  
  
}

package com.javferna.packtpub.mastering.knn.loader;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.nio.file.Paths;  
import java.util.ArrayList;  
import java.util.List;  
  
import com.javferna.packtpub.mastering.knn.data.BankMarketing;  
  
*/\*\*  
 \* Class that loads the examples of the Bank Marketing data set from a file  
 \** ***@author*** *author  
 \*  
 \*/*public class BankMarketingLoader {  
  
 */\*\*  
 \* Method that loads the examples of the Bank Marketing data set from a file  
 \** ***@param*** *dataPath Path to the file where the data items are stored  
 \** ***@return*** *List of BankMarketing examples  
 \*/* public List<BankMarketing> load (String dataPath) {  
 Path file=Paths.*get*(dataPath);  
 List<BankMarketing> dataSet=new ArrayList<>();  
 try (InputStream in = Files.*newInputStream*(file);  
 BufferedReader reader =  
 new BufferedReader(new InputStreamReader(in))) {  
 String line = null;  
 while ((line = reader.readLine()) != null) {  
 String data[]=line.split(";");  
 BankMarketing dataObject=new BankMarketing();  
 dataObject.setData(data);  
 dataSet.add(dataObject);  
 }  
 } catch (IOException x) {  
 x.printStackTrace();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return dataSet;  
 }  
}

package com.javferna.packtpub.mastering.knn.main;  
  
import java.util.Date;  
import java.util.List;  
  
import com.javferna.packtpub.mastering.knn.data.BankMarketing;  
import com.javferna.packtpub.mastering.knn.loader.BankMarketingLoader;  
import com.javferna.packtpub.mastering.knn.parallel.group.KnnClassifierParallelGroup;  
  
*/\*\*  
 \* Main class that launches the tests using the coarse-grained concurrent version and serial sorting  
 \** ***@author*** *author  
 \*  
 \*/*public class ParallelGrouplMain {  
  
 public static void main(String[] args) {  
  
 BankMarketingLoader loader = new BankMarketingLoader();  
 List<BankMarketing> train = loader.load("data\\bank.data");  
 System.*out*.println("Train: " + train.size());  
 List<BankMarketing> test = loader.load("data\\bank.test");  
 System.*out*.println("Test: " + test.size());  
 double currentTime = 0.0d;  
 int success = 0, mistakes = 0;  
   
 int k = 10;  
 if (args.length==1) {  
 k = Integer.*parseInt*(args[0]);  
 }  
   
 success = 0;  
 mistakes = 0;  
 KnnClassifierParallelGroup classifier = new KnnClassifierParallelGroup(  
 train, k, 1, false);  
 try {  
 Date start, end;  
 start = new Date();  
 for (BankMarketing example : test) {  
 String tag = classifier.classify(example);  
 if (tag.equals(example.getTag())) {  
 success++;  
 } else {  
 mistakes++;  
 }  
 }  
 end = new Date();  
  
 currentTime = end.getTime() - start.getTime();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 classifier.destroy();  
  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.*out*.println("Parallel Classifier Group - K: " + k  
 + " - Factor 1 - Parallel Sort: false");  
 System.*out*.println("Success: " + success);  
 System.*out*.println("Mistakes: " + mistakes);  
 System.*out*.println("Execution Time: " + (currentTime / 1000)  
 + " seconds.");  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
  
 }  
  
}

package com.javferna.packtpub.mastering.knn.main;  
  
import java.util.Date;  
import java.util.List;  
  
import com.javferna.packtpub.mastering.knn.data.BankMarketing;  
import com.javferna.packtpub.mastering.knn.loader.BankMarketingLoader;  
import com.javferna.packtpub.mastering.knn.parallel.group.KnnClassifierParallelGroup;  
  
*/\*\*  
 \* Main class that launches the tests using the coarse-grained concurrent  
 \* version and serial sorting  
 \*   
 \** ***@author*** *author  
 \*  
 \*/*public class ParallelGroupMainSort {  
  
 public static void main(String[] args) {  
  
 BankMarketingLoader loader = new BankMarketingLoader();  
 List<BankMarketing> train = loader.load("data\\bank.data");  
 System.*out*.println("Train: " + train.size());  
 List<BankMarketing> test = loader.load("data\\bank.test");  
 System.*out*.println("Test: " + test.size());  
 double currentTime = 0.0d;  
 int success = 0, mistakes = 0;  
  
 int k = 10;  
 if (args.length==1) {  
 k = Integer.*parseInt*(args[0]);  
 }  
   
 success = 0;  
 mistakes = 0;  
 KnnClassifierParallelGroup classifier = new KnnClassifierParallelGroup(train, k, 1, true);  
 try {  
 Date start, end;  
 start = new Date();  
 for (BankMarketing example : test) {  
 String tag = classifier.classify(example);  
 if (tag.equals(example.getTag())) {  
 success++;  
 } else {  
 mistakes++;  
 }  
 }  
 end = new Date();  
  
 currentTime = end.getTime() - start.getTime();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 classifier.destroy();  
  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.*out*.println("Parallel Classifier Group - K: " + k + " - Factor 1 - Parallel Sort: true");  
 System.*out*.println("Success: " + success);  
 System.*out*.println("Mistakes: " + mistakes);  
 System.*out*.println("Execution Time: " + (currentTime / 1000) + " seconds.");  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
  
 }  
  
}

package com.javferna.packtpub.mastering.knn.parallel.group;  
  
import java.util.List;  
import java.util.concurrent.CountDownLatch;  
  
import com.javferna.packtpub.mastering.knn.data.Distance;  
import com.javferna.packtpub.mastering.knn.data.Sample;  
import com.javferna.packtpub.mastering.knn.distances.EuclideanDistanceCalculator;  
  
*/\*\*  
 \* Task that of the coarse-grained concurrent version   
 \** ***@author*** *author  
 \*  
 \*/*public class GroupDistanceTask implements Runnable {  
  
 */\*\*  
 \* Array of distances  
 \*/* private final Distance[] distances;  
   
 */\*\*  
 \* Indexes that determines the examples of the train data this task will process  
 \*/* private final int startIndex, endIndex;  
   
 */\*\*  
 \* Example of the test data we want to classify  
 \*/* private final Sample example;  
   
 */\*\*  
 \* Data set with the train data examples  
 \*/* private final List<? extends Sample> dataSet;  
   
 */\*\*  
 \* Synchronization mechanism to control the end of the task  
 \*/* private final CountDownLatch endControler;  
  
 */\*\*  
 \* Constructor of the class. Initializes all the internal data  
 \** ***@param*** *distances Array of distances  
 \** ***@param*** *startIndex Start index that determines the examples of the train data this task will process  
 \** ***@param*** *endIndex End index that determines the examples of the train data this task will process  
 \** ***@param*** *dataSet Data set with the train data examples  
 \** ***@param*** *example Example of the test data we want to classify  
 \** ***@param*** *endControler Synchronization mechanism to control the end of the task  
 \*/* public GroupDistanceTask(Distance[] distances, int startIndex,  
 int endIndex, List<? extends Sample> dataSet, Sample example,  
 CountDownLatch endControler) {  
 this.distances = distances;  
 this.startIndex = startIndex;  
 this.endIndex = endIndex;  
 this.example = example;  
 this.dataSet = dataSet;  
 this.endControler = endControler;  
 }  
  
 @Override  
 */\*\*  
 \* Concurrent task that calculates the distance between the example and the train instances between  
 \* the startIndex and the endIndex   
 \*/* public void run() {  
 for (int index = startIndex; index < endIndex; index++) {  
 Sample localExample=dataSet.get(index);  
 distances[index] = new Distance();  
 distances[index].setIndex(index);  
 distances[index].setDistance(EuclideanDistanceCalculator  
 .*calculate*(localExample, example));  
 }  
 endControler.countDown();  
 }  
  
}

package com.javferna.packtpub.mastering.knn.parallel.group;  
  
import java.util.Arrays;  
import java.util.Collections;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
import java.util.concurrent.CountDownLatch;  
import java.util.concurrent.Executors;  
import java.util.concurrent.ThreadPoolExecutor;  
  
import com.javferna.packtpub.mastering.knn.data.Distance;  
import com.javferna.packtpub.mastering.knn.data.Sample;  
import com.javferna.packtpub.mastering.knn.distances.EuclideanDistanceCalculator;  
  
*/\*\*  
 \* Coarse-grained concurrent version of the Knn algorithm  
 \** ***@author*** *author  
 \*  
 \*/*public class KnnClassifierParallelGroup {  
  
 */\*\*  
 \* Train data  
 \*/* private List<? extends Sample> dataSet;  
   
 */\*\*  
 \* K parameter  
 \*/* private int k;  
   
 */\*\*  
 \* Executor to execute the concurrent tasks  
 \*/* private ThreadPoolExecutor executor;  
   
 */\*\*  
 \* Number of threads to configure the executor  
 \*/* private int numThreads;  
   
 */\*\*  
 \* Check to indicate if we use the serial or the parallel sorting  
 \*/* private boolean parallelSort;  
   
 */\*\*  
 \* Constructor of the class. Initialize the internal data  
 \** ***@param*** *dataSet Train data set  
 \** ***@param*** *k K parameter  
 \** ***@param*** *factor Factor of increment of the number of cores  
 \** ***@param*** *parallelSort Check to indicate if we use the serial or the parallel sorting  
 \*/* public KnnClassifierParallelGroup(List<? extends Sample> dataSet, int k, int factor, boolean parallelSort) {  
 this.dataSet=dataSet;  
 this.k=k;  
 numThreads=factor\*(Runtime.*getRuntime*().availableProcessors());  
 numThreads=2;  
 executor=(ThreadPoolExecutor) Executors.*newFixedThreadPool*(numThreads);  
 this.parallelSort=parallelSort;  
 }  
   
 */\*\*  
 \* Method that classify an example  
 \** ***@param*** *example Example to classify  
 \** ***@return*** *Class or tag of the example  
 \** ***@throws*** *Exception Exception if something goes wrong  
 \*/* public String classify (Sample example) throws Exception {  
   
 Distance[] distances=new Distance[dataSet.size()];  
 CountDownLatch endControler=new CountDownLatch(numThreads);  
   
 int length=dataSet.size()/numThreads;  
 int startIndex=0, endIndex=length;  
   
 for (int i=0; i <numThreads; i++) {  
// GroupDistanceTask task=new GroupDistanceTask(distances, startIndex, endIndex, dataSet, example, endControler);  
  
 int finalStartIndex = startIndex;  
 int finalEndIndex = endIndex;  
 Runnable task = () -> {  
 for (int index = finalStartIndex; index < finalEndIndex; index++) {  
 Sample localExample=dataSet.get(index);  
 distances[index] = new Distance();  
 distances[index].setIndex(index);  
 distances[index].setDistance(EuclideanDistanceCalculator  
 .*calculate*(localExample, example));  
 }  
 endControler.countDown();  
 };  
  
 startIndex=endIndex;  
 if (i<numThreads-2) {  
 endIndex=endIndex+length;   
 } else {  
 endIndex=dataSet.size();  
 }  
 executor.execute(task);  
   
 }  
 endControler.await();  
  
 if (parallelSort) {  
 Arrays.*parallelSort*(distances);  
 } else {  
 Arrays.*sort*(distances);  
 }  
   
 Map<String, Integer> results = new HashMap<>();  
 for (int i = 0; i < k; i++) {  
 Sample localExample = dataSet.get(distances[i].getIndex());  
 String tag = localExample.getTag();  
 results.merge(tag, 1, (a, b) -> a+b);  
 }  
   
 return Collections.*max*(results.entrySet(),  
 Map.Entry.*comparingByValue*()).getKey();  
 }  
   
 */\*\*  
 \* Method that finish the execution of the executor  
 \*/* public void destroy() {  
 executor.shutdown();  
 }  
   
}

**Serial Execution Time:** 72.73 seconds

**Parallel Execution Time run on the 4-core intel core i7 processor with RAM 8GB.**

|  |  |
| --- | --- |
| **No of Thread** | **Execution Time** |
| 1 | 69.215 |
| 2 | 68.782 |
| 3 | 68.389 |
| 4 | 65.434 |