Laporan Tugas Besar 1
IF4071 Pembelajaran Mesin

Oleh :

Muhammad Nizami	/ 13512501
Ivan Andrianto	/ 13513039
Wilhelmus Andrian	/ 13513071

Sekolah Tinggi Elektro dan Informatika
Informatika
Institut Teknologi Bandung

2016

1. Source Code

```
Filename : CsvToArff.java

public class CsvToArff {
    public CsvToArff(String filename) throws IOException{
        // load CSV
        CSVLoader loader = new CSVLoader();
        URL url = getClass().getResource("../"+filename);
        loader.setSource(new File(url.getPath()));
        Instances data = loader.getDataSet();

        // save ARFF
        ArffSaver saver = new ArffSaver();
        saver.setInstances(data);
        String newfile = "src/"+filename.substring(0,
filename.lastIndexOf('.'))+".arff";
        saver.setFile(new File(newfile));
        saver.writeBatch();
    }
}
```

```
Filename : CustomFilter.java

public class CustomFilter {
    public CustomFilter(){

    }

    public Instances removeAttribute(Instances structure) throws Exception{
        //NORMALIZE AND REMOVE USELESS ATTRIBUTES
        Normalize norm = new Normalize();
        norm.setInputFormat(structure);
        structure = Filter.useFilter(structure, norm);

        RemoveUseless ru = new RemoveUseless();
        ru.setInputFormat(structure);
        structure = Filter.useFilter(structure, ru);

        Ranker rank = new Ranker();
        InfoGainAttributeEval eval = new InfoGainAttributeEval();
        eval.buildEvaluator(structure);
        //END OF NORMALIZATION

        return structure;
    }

    public Instances resampling(Instances structure){
        Resample filter = new Resample();
      Instances filteredIns = null;
      filter.setBiasToUniformClass(1.0);
      try {
            filter.setInputFormat(structure);
            filter.setNoReplacement(false);
            filter.setSampleSizePercent(100);
            filteredIns = Filter.useFilter(structure, filter);
      } catch (Exception e) {
            e.printStackTrace();
      }
      return filteredIns;
    }

    public Instances convertNumericToNominal(Instances structure) throws
Exception {
        NumericToNominal convert= new NumericToNominal();
        String[] options= new String[2];
        options[0]="-R";
        options[1]= "1-" + structure.numAttributes();
        convert.setOptions(options);
        convert.setInputFormat(structure);
        structure = Filter.useFilter(structure, convert);
        return structure;
    }

    public Instances convertNumericRange(Instances structure) throws
Exception{
        for(int i = 0; i<structure.numAttributes()-1;i++){

if(structure.attribute(i).typeToString(structure.attribute(i)).equals("numeri
")){
                structure.sort(i);
                structure = toRange(structure,i);
            }
        }
        return structure;
    }

    //SET ALL VALUES TO THE BATAS BAWAH
    private Instances toRange(Instances structure,int index)throws Exception{
        Attribute attr = structure.attribute(index);
        Attribute classlabel =
structure.attribute(structure.numAttributes()-1);
        String label = structure.instance(0).stringValue(classlabel);
        double threshold = structure.instance(0).value(index);
        for(int i = 0; i<structure.numInstances();i++){
            if(!structure.instance(i).stringValue(classlabel).equals(label)){
                label = structure.instance(i).stringValue(classlabel);
                threshold = structure.instance(i).value(index);
            }
            structure.instance(i).setValue(attr, threshold);
        }
        return structure;
    }
    public String convertToFit(String value, Instances data, int index){
        int i;
        String threshold = data.attribute(index).value(0);
        for(i = 0; i < data.numDistinctValues(data.attribute(index)) ;i++){

if(Float.valueOf(value)<Float.valueOf(data.attribute(index).value(i))){
                value = threshold;
                return value;
            }
            threshold = data.attribute(index).value(i);
        }
        value = threshold;
        return value;
    }

}
```

```
Filename : Model.java

public class Model {

    public static void save(Classifier cls, String path) throws IOException {
        ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(path));
        oos.writeObject(cls);
        oos.flush();
        oos.close();
    }

    public static Classifier load(String path) throws IOException,
ClassNotFoundException {
        ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(path));
        Classifier cls = (Classifier) ois.readObject();
        ois.close();
        return cls;
    }

}
```

```
Filename : MyId3.java
public class myID3 extends Classifier implements Serializable {

    private class TreeNode implements Serializable{
        public Attribute decision;
        public double branchValue;
        public int label;
        public List<TreeNode> branches;
        public TreeNode(){
            decision = null;
            label = -1;
            branchValue = -1;
            branches = new LinkedList<>();
        }
        public void addBranch(TreeNode node){
            branches.add(node);
        }
        public boolean isLeafNode(){
            return label >= 0;
        }
        public String stringify(int tabLevel){
            String retval = "";
            for (int i=0;i<tabLevel;i++){
                retval+='\t';
            }
            retval+=branchValue;
            retval+="-";
            if (isLeafNode())
                retval+=label;
            else
                retval+=decision.name();
            retval+='\n';
            for (TreeNode branch : branches){
                retval+=branch.stringify(tabLevel+1);
            }
            return retval;
        }
    }

    private TreeNode rootNode;

    public TreeNode id3Node(Instances i){
        TreeNode treeNode = new TreeNode();

        int [] count = calculateCount(i);
        for (int j=0;j<count.length;j++){
            int c = count[j];
            if (c == i.numInstances()){
               treeNode.label = j;
               return treeNode;
            }
        }

        if (i.numAttributes()<=1){
            int maxc = -1;
            int maxcj = -1;
            for (int j=0;j<count.length;j++){
                if (count[j]>maxc){
                    maxc=count[j];
                    maxcj = j;
                }
            }
            treeNode.label = maxcj;
            return treeNode;
        }

        Attribute bestA = null;
        double bestAIG = -1;
        double entropyOfSet = entropy(i);
        for (int j=0;j<i.numAttributes();j++){
            Attribute a = i.attribute(j);
            if (a!=i.classAttribute()){
                double aIG = infoGain(i,a,entropyOfSet);
                if (aIG>bestAIG){
                    bestAIG = aIG;
                    bestA = a;
                }
            }
        }
        treeNode.decision = bestA;
        Instances [] subSets = splitData(i, bestA);
        for (Instances subSet : subSets){
            if (subSet.numInstances()>0){
                double attributeValue = subSet.firstInstance().value(bestA);
                subSet.deleteAttributeAt(bestA.index());
                TreeNode newBranch = id3Node(subSet);
                newBranch.branchValue = attributeValue;
                treeNode.addBranch(newBranch);
            }else{
            }
        }
        return treeNode;
    }
    private Instances[] splitData(Instances data, Attribute att) {

        Instances[] splitData = new Instances[att.numValues()];
        for (int j = 0; j < att.numValues(); j++) {
            splitData[j] = new Instances(data, data.numInstances());
        }
        Enumeration instEnum = data.enumerateInstances();
        while (instEnum.hasMoreElements()) {
            Instance inst = (Instance) instEnum.nextElement();
            splitData[(int) inst.value(att)].add(inst);
        }
        for (int i = 0; i < splitData.length; i++) {
            splitData[i].compactify();
        }
        return splitData;
      }

    @Override
    public void buildClassifier(Instances i) throws Exception {
        rootNode = id3Node(i);
    }
    @Override
    public double classifyInstance(Instance instance){
        TreeNode nodeIter = rootNode;
        while (!nodeIter.isLeafNode()){
            double decisionAttrVal = instance.value(nodeIter.decision);
            boolean branchFound = false;
            for (TreeNode branch : nodeIter.branches){
                if (branch.branchValue==decisionAttrVal){
                    nodeIter = branch;
                    branchFound = true;
                    break;
                }
            }
            if (!branchFound){
                return 0;
            }
        }
        return nodeIter.label;
    }

    public double calculateAttributeProportion(Instances instances, Attribute
attribute) {
        int numClasses = instances.classAttribute().numValues();

        return 0;
    }

    public myID3() {
    }

    public HashMap<String, Integer> getAttributeValues(Instances instances,
Attribute attribute) {
        int numInstances = instances.numInstances();
        HashMap<String, Integer> values = new HashMap<String, Integer>();
        for (int i = 0; i < numInstances; i++) {
            String key = instances.instance(i).stringValue(attribute);
            if (values.containsKey(key)) {
                values.put(key, values.get(key) + 1);
            } else {
                values.put(key, 1);
            }
        }
        return values;
    }

    private Instances filterInstanceWithAttributeValue(Instances instances,
Attribute attribute, String value) {
        Instances newInstances = new Instances(instances);
        newInstances.delete();
        int numInstances = instances.numInstances();
        for (int i = 0; i < numInstances; i++) {
            Instance instance = instances.instance(i);
            if (instance.stringValue(attribute).equals(value)) {
                newInstances.add(instance);
            }
        }
        return newInstances;
    }

    public double infoGain(Instances instances, Attribute attribute, double
entropyOfSet){
        int numClasses = instances.classAttribute().numValues();
        int numInstances = instances.numInstances();
        HashMap<String, Integer> values = getAttributeValues(instances,
attribute);
        double zigma = 0;
        Set<String> keys = values.keySet();
        for (int i = 0; i < keys.size(); i++) {
            String key = nthElement(keys, i);
            Instances instanceWithAttributeValue =
filterInstanceWithAttributeValue(instances, attribute, key);
            zigma += (values.get(key) / numInstances) *
entropy(instanceWithAttributeValue);
        }
        return entropyOfSet - zigma;
    }

    public double entropy(Instances instances){
        double result = 0;
        double[] proportion = calculateProportion(instances);
        for (double p : proportion){
            if (p!=0)
                result -= Math.log(p)/Math.log(2)*p;
        }
        return result;
    }

    public int [] calculateCount(Instances instances){

        int numClasses = instances.classAttribute().numValues();
        int numInstances = instances.numInstances();
        int [] num = new int[numClasses];
        for (int i=0;i<numClasses;i++){
            num[i]=0;
        }
        for (int i=0;i<numInstances;i++){
            Instance instance = instances.instance(i);
            double classValue = instance.value(instance.classAttribute());
            int classIndex = (int) classValue;
            num[classIndex]++;
        }
        return num;
    }

    public double [] calculateProportion(Instances instances){
        int numClasses = instances.classAttribute().numValues();
        int numInstances = instances.numInstances();
        int [] num = calculateCount(instances);
        double [] result = new double[numClasses];
        for (int i=0;i<numClasses;i++){
            result[i] = ((double)num[i])/numInstances;
        }
        return result;
    }

    public static final <T> T nthElement(Iterable<T> data, int n){
    int index = 0;
    for(T element : data){
        if(index == n){
            return element;
        }
        index++;
      }
    return null;
  }

    public String toString(){
        return rootNode.stringify(0);
    }
}
```

Filename : MyC45.java

```
public class myC45 extends Classifier implements Serializable{

    public myC45(){

    };
    protected PruneableClassifierTree m_root;

    private Instances[] splitData(Instances data, Attribute att) {

        Instances[] splitData = new Instances[att.numValues()];
        for (int j = 0; j < att.numValues(); j++) {
            splitData[j] = new Instances(data, data.numInstances());
        }
        Enumeration instEnum = data.enumerateInstances();
        while (instEnum.hasMoreElements()) {
            Instance inst = (Instance) instEnum.nextElement();
            splitData[(int) inst.value(att)].add(inst);
        }
        for (int i = 0; i < splitData.length; i++) {
            splitData[i].compactify();
        }
        return splitData;
    }

    private class TreeNode implements Serializable{
        public Attribute decision;
        public double branchValue;
        public int label;
        public List<TreeNode> branches;
        public TreeNode(){
            decision = null;
            label = -1;
            branchValue = -1;
            branches = new LinkedList<>();
        }
        public void addBranch(TreeNode node){
            branches.add(node);
        }
        public boolean isLeafNode(){
            return label >= 0;
        }
        public String stringify(int tabLevel){
            String retval = "";
            for (int i=0;i<tabLevel;i++){
                retval+='\t';
            }
            retval+=branchValue;
            retval+="-";
            if (isLeafNode())
                retval+=label;
            else
                retval+=decision.name();
            retval+='\n';
            for (TreeNode branch : branches){
                retval+=branch.stringify(tabLevel+1);
            }
            return retval;
        }
    }

    private TreeNode rootNode;

    private int vote(Instances data, Attribute att){
        int [] count = calculateCount(data);
        int maxc = -1;
        int maxcj = -1;
        for (int j=0;j<count.length;j++){
            if (count[j]>maxc){
                maxc=count[j];
                maxcj = j;
            }
        }
        return maxcj;
    }
    public TreeNode C45Node(Instances i, double parentGain){
        TreeNode treeNode = new TreeNode();

        int [] count = calculateCount(i);
        for (int j=0;j<count.length;j++){
            int c = count[j];
            if (c == i.numInstances()){
               treeNode.label = j;
               return treeNode;
            }
        }

        if (i.numAttributes()<=1){
            int maxc = -1;
            int maxcj = -1;
            for (int j=0;j<count.length;j++){
                if (count[j]>maxc){
                    maxc=count[j];
                    maxcj = j;
                }
            }
            treeNode.label = maxcj;
            return treeNode;
        }

        Attribute bestA = null;
        double bestAIG = -1;
        double entropyOfSet = entropy(i);
        for (int j=0;j<i.numAttributes();j++){
            Attribute a = i.attribute(j);
            if (a!=i.classAttribute()){
                double aIG = infoGain(i,a,entropyOfSet);
                if (aIG>bestAIG){
                    bestAIG = aIG;
                    bestA = a;
                }
            }
        }
        double childGainRatio = gainRatio(bestAIG,entropyOfSet);
        treeNode.decision = bestA;
        if(childGainRatio > parentGain){
            Instances [] subSets = splitData(i, bestA);
            for (Instances subSet : subSets){
                if (subSet.numInstances()>0){
                    double attributeValue =
subSet.firstInstance().value(bestA);
                    subSet.deleteAttributeAt(bestA.index());
                    TreeNode newBranch = C45Node(subSet,childGainRatio);
                    newBranch.branchValue = attributeValue;
                    treeNode.addBranch(newBranch);
                }
            }
        }
        else{
            TreeNode newBranch = new TreeNode();
            newBranch.label = vote(i,bestA);
            newBranch.branchValue = treeNode.branchValue;
            treeNode.addBranch(newBranch);
        }
        return treeNode;
    }

    @Override
    public void buildClassifier(Instances i) throws Exception {
        rootNode = C45Node(i,-1.0);
    }

    @Override
    public double classifyInstance(Instance instance){
        TreeNode nodeIter = rootNode;
        while (!nodeIter.isLeafNode()){
            double decisionAttrVal = instance.value(nodeIter.decision);
            boolean branchFound = false;
            for (TreeNode branch : nodeIter.branches){
                if (branch.branchValue==decisionAttrVal){
                    nodeIter = branch;
                    branchFound = true;
                    break;
                }
            }
            if (!branchFound){
                return 0;
            }
        }
        return nodeIter.label;
    }

    public HashMap<String, Integer> getAttributeValues(Instances instances,
Attribute attribute) {
        int numInstances = instances.numInstances();
        HashMap<String, Integer> values = new HashMap<String, Integer>();
        for (int i = 0; i < numInstances; i++) {
            String key = instances.instance(i).stringValue(attribute);
            if (values.containsKey(key)) {
                values.put(key, values.get(key) + 1);
            } else {
                values.put(key, 1);
            }
        }
        return values;
    }

    private Instances filterInstanceWithAttributeValue(Instances instances,
Attribute attribute, String value) {
        Instances newInstances = new Instances(instances);
        newInstances.delete();
        int numInstances = instances.numInstances();
        for (int i = 0; i < numInstances; i++) {
            Instance instance = instances.instance(i);
            if (instance.stringValue(attribute).equals(value)) {
                newInstances.add(instance);
            }
        }
        return newInstances;
    }

    public double infoGain(Instances instances, Attribute attribute, double
entropyOfSet){
        int numInstances = instances.numInstances();
        HashMap<String, Integer> values = getAttributeValues(instances,
attribute);
        double zigma = 0;
        Set<String> keys = values.keySet();
        for (int i = 0; i < keys.size(); i++) {
            String key = nthElement(keys, i);
            Instances instanceWithAttributeValue =
filterInstanceWithAttributeValue(instances, attribute, key);
            zigma += (values.get(key) / numInstances) *
entropy(instanceWithAttributeValue);
        }
        return entropyOfSet - zigma;
    }

    public double gainRatio(double gain, double entropyOfSet){
        return gain/entropyOfSet;
    }

    public double entropy(Instances instances){
        double result = 0;
        double[] proportion = calculateProportion(instances);
        for (double p : proportion){
            if (p!=0)
                result -= Math.log(p)/Math.log(2)*p;
        }
        return result;
    }

    public int [] calculateCount(Instances instances){

        int numClasses = instances.classAttribute().numValues();
        int numInstances = instances.numInstances();
        int [] num = new int[numClasses];
        for (int i=0;i<numClasses;i++){
            num[i]=0;
        }
        for (int i=0;i<numInstances;i++){
            Instance instance = instances.instance(i);
            double classValue = instance.value(instance.classAttribute());
            int classIndex = (int) classValue;
            num[classIndex]++;
        }
        return num;
    }

    public double [] calculateProportion(Instances instances){
        int numClasses = instances.classAttribute().numValues();
        int numInstances = instances.numInstances();
        int [] num = calculateCount(instances);
        double [] result = new double[numClasses];
        for (int i=0;i<numClasses;i++){
            result[i] = ((double)num[i])/numInstances;
        }
        return result;
    }

    public static final <T> T nthElement(Iterable<T> data, int n){
    int index = 0;
    for(T element : data){
        if(index == n){
            return element;
        }
        index++;
      }
    return null;
  }

    public String toString(){
        return rootNode.stringify(0);
    }

}
```

Filename : Main.java

```
public class Main {

    public static boolean isNumeric(String s) {
        return s.matches("[-+]?\\d*\\.?\\d+");
    }
    public static BufferedReader readDataFile(String filename) {
        BufferedReader inputReader = null;
        try {
                inputReader = new BufferedReader(new FileReader(filename));
        } catch (FileNotFoundException ex) {
                System.err.println("File not found: " + filename);
        }
        return inputReader;
    }
    public static Instances[][] crossValidationSplit(Instances data, int
numberOfFolds) {
        Instances[][] split = new Instances[2][numberOfFolds];
        for (int i = 0; i < numberOfFolds; i++) {
                split[0][i] = data.trainCV(numberOfFolds, i);
                split[1][i] = data.testCV(numberOfFolds, i);
        }
        return split;
    }

    public static Evaluation classify(Classifier model,Instances trainingSet,
Instances testingSet) throws Exception {
        Evaluation evaluation = new Evaluation(trainingSet);
        model.buildClassifier(trainingSet);
        evaluation.evaluateModel(model, testingSet);
        return evaluation;
    }

    public static double calculateAccuracy(FastVector predictions) {
        double correct = 0;
        for (int i = 0; i < predictions.size(); i++) {
                NominalPrediction np = (NominalPrediction)
predictions.elementAt(i);
                if (np.predicted() == np.actual()) {
                        correct++;
                }
        }
        return 100 * correct / predictions.size();
    }


    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws IOException, Exception {
        // TODO code application logic here
        String filename = "weather";

        //Masih belum mengerti tipe .csv yang dapat dibaca seperti apa
        //CsvToArff convert = new CsvToArff(filename+".csv");

        //LOAD FILE
        BufferedReader datafile = readDataFile("src/"+filename+".arff");
        Instances data = new Instances(datafile);
        data.setClassIndex(data.numAttributes()-1);
        //END OF LOAD FILE

        CustomFilter fil = new CustomFilter();

        //REMOVE USELESS ATTRIBUTE
        data = fil.removeAttribute(data);
        System.out.println(data);
        Instances[] allData = new Instances[4];
        //data for Id3
        allData[0] = fil.resampling(fil.convertNumericToNominal(data));
        //data for J48
        allData[1] = fil.convertNumericToNominal(fil.resampling(data));
        //data for myId3
        allData[2] = allData[0];
        //data for myC4.5
        allData[3] =
fil.resampling(fil.convertNumericToNominal(fil.convertNumericRange(data)));

        data = fil.convertNumericToNominal(data);
        // BUILD CLASSIFIERS
        Classifier[] models = {
            new Id3(), //C4.5
            new J48(),
            new myID3(),
            new myC45()
        };

        for (int j = 0; j < models.length; j++) {
            FastVector predictions = new FastVector();
            //FOR TEN-FOLD CROSS VALIDATION
            Instances[][] split = crossValidationSplit(allData[j], 10);
            // Separate split into training and testing arrays
            Instances[] trainingSplits = split[0];
            Instances[] testingSplits = split[1];

            for (int i = 0; i < trainingSplits.length; i++) {
                try {
                    System.out.println("Building for training Split : " + i);
                    Evaluation validation = classify(models[j],
trainingSplits[i], testingSplits[i]);

                    predictions.appendElements(validation.predictions());

                    // Uncomment to see the summary for each training-testing
pair.
//                    System.out.println(models[j].toString());
                } catch (Exception ex) {
                    Logger.getLogger(Main.class.getName()).log(Level.SEVERE,
null, ex);
                }
                // Calculate overall accuracy of current classifier on all
splits
                double accuracy = calculateAccuracy(predictions);

                // Print current classifier's name and accuracy in a
complicated,
                // but nice-looking way.
                System.out.println("Accuracy of " +
models[j].getClass().getSimpleName() + ": "
```

```
                              + String.format("%.2f%%", accuracy)
                              + "\n----------------------------------");
            }
            models[j].buildClassifier(allData[j]);
            Model.save(models[j],models[j].getClass().getSimpleName());
        }


        //test instance
        Instances trainingSet = new Instances("Rel",
getFvWekaAttributes(data), 10);
        trainingSet.setClassIndex(data.numAttributes() - 1);

        Instance testInstance = new Instance(data.numAttributes());
        for(int i = 0; i<data.numAttributes()-1;i++){
            System.out.print("Masukkan "+ data.attribute(i).name()+" : ");
            Scanner in = new Scanner(System.in);
            String att = in.nextLine();
            if(isNumeric(att)){
                att = fil.convertToFit(att, data, i);
            }
            testInstance.setValue(data.attribute(i),att);
        }

//        System.out.println(testInstance);
//        System.out.println(testInstance.classAttribute().index());


        trainingSet.add(testInstance);

        Classifier Id3 = Model.load("Id3");
        Classifier J48 = Model.load("J48");
        Classifier myID3 = Model.load("myID3");
        Classifier MyC45 = Model.load("myC45");
//        Classifier MyId3 = Model.load("myID3");

        Instance A = trainingSet.instance(0);
        Instance B = trainingSet.instance(0);
        Instance C = trainingSet.instance(0);
        Instance D = trainingSet.instance(0);

//test with ID3 WEKA
        A.setClassValue(Id3.classifyInstance(trainingSet.instance(0)));
        System.out.println("Id3 Weka : " + A);

        //test with C4.5 WEKA
        B.setClassValue(J48.classifyInstance(trainingSet.instance(0)));
        System.out.println("C4.5 Weka : " + B);

        //test with my C4.5
        C.setClassValue(MyC45.classifyInstance(trainingSet.instance(0)));
        System.out.println("My C4.5 : " + C);
        //test with my ID3
        D.setClassValue(myID3.classifyInstance(trainingSet.instance(0)));
        System.out.println("My ID3 : " + D);
    }

    private static FastVector getFvWekaAttributes(Instances data) {
        int numAttributes = data.numAttributes();
        FastVector fvWekaAttributes = new FastVector(numAttributes);
        for (int i = 0; i < numAttributes; i++) {
            fvWekaAttributes.addElement(data.attribute(i));
        }
        return fvWekaAttributes;
    }

}
```

2. **Hasil Eksekusi terhadap data tes**
    a. **Data weather (binary class)**

```
Hasil 10-fold cross validation pada 4 metode:
Building for training Split : 0
Accuracy of Id3: 50.00%
---------------------------------
Building for training Split : 1
Accuracy of Id3: 50.00%
---------------------------------
Building for training Split : 2
Accuracy of Id3: 66.67%
---------------------------------
Building for training Split : 3
Accuracy of Id3: 75.00%
---------------------------------
Building for training Split : 4
Accuracy of Id3: 77.78%
---------------------------------
Building for training Split : 5
Accuracy of Id3: 80.00%
---------------------------------
Building for training Split : 6
Accuracy of Id3: 72.73%
---------------------------------
Building for training Split : 7
Accuracy of Id3: 75.00%
---------------------------------
Building for training Split : 8
Accuracy of Id3: 76.92%
---------------------------------
Building for training Split : 9
```

```
Accuracy of Id3: 71.43%
---------------------------------
Building for training Split : 0
Accuracy of J48: 50.00%
---------------------------------
Building for training Split : 1
Accuracy of J48: 75.00%
---------------------------------
Building for training Split : 2
Accuracy of J48: 83.33%
---------------------------------
Building for training Split : 3
Accuracy of J48: 87.50%
---------------------------------
Building for training Split : 4
Accuracy of J48: 88.89%
---------------------------------
Building for training Split : 5
Accuracy of J48: 90.00%
---------------------------------
Building for training Split : 6
Accuracy of J48: 81.82%
---------------------------------
Building for training Split : 7
Accuracy of J48: 83.33%
---------------------------------
Building for training Split : 8
Accuracy of J48: 84.62%
---------------------------------
Building for training Split : 9
Accuracy of J48: 85.71%
---------------------------------
Building for training Split : 0
Accuracy of myID3: 100.00%
---------------------------------
Building for training Split : 1
Accuracy of myID3: 75.00%
---------------------------------
Building for training Split : 2
Accuracy of myID3: 66.67%
---------------------------------
Building for training Split : 3
Accuracy of myID3: 62.50%
---------------------------------
Building for training Split : 4
Accuracy of myID3: 66.67%
---------------------------------
```

```
Building for training Split : 5
Accuracy of myID3: 70.00%
---------------------------------
Building for training Split : 6
Accuracy of myID3: 63.64%
---------------------------------
Building for training Split : 7
Accuracy of myID3: 66.67%
---------------------------------
Building for training Split : 8
Accuracy of myID3: 69.23%
---------------------------------
Building for training Split : 9
Accuracy of myID3: 71.43%
---------------------------------
Building for training Split : 0
Accuracy of myC45: 100.00%
---------------------------------
Building for training Split : 1
Accuracy of myC45: 75.00%
---------------------------------
Building for training Split : 2
Accuracy of myC45: 66.67%
---------------------------------
Building for training Split : 3
Accuracy of myC45: 62.50%
---------------------------------
Building for training Split : 4
Accuracy of myC45: 66.67%
---------------------------------
Building for training Split : 5
Accuracy of myC45: 70.00%
---------------------------------
Building for training Split : 6
Accuracy of myC45: 63.64%
---------------------------------
Building for training Split : 7
Accuracy of myC45: 66.67%
---------------------------------
Building for training Split : 8
Accuracy of myC45: 69.23%
---------------------------------
Building for training Split : 9
Accuracy of myC45: 71.43%
---------------------------------
```

b. **Data iris (multi-class)**

```
Hasil 10-fold cross validation pada 4 metode:
Building for training Split : 0
Accuracy of Id3: 93.33%
---------------------------------
Building for training Split : 1
Accuracy of Id3: 96.67%
---------------------------------
Building for training Split : 2
Accuracy of Id3: 88.89%
---------------------------------
Building for training Split : 3
Accuracy of Id3: 90.00%
---------------------------------
Building for training Split : 4
Accuracy of Id3: 90.67%
---------------------------------
Building for training Split : 5
Accuracy of Id3: 92.22%
---------------------------------
Building for training Split : 6
Accuracy of Id3: 92.38%
---------------------------------
Building for training Split : 7
Accuracy of Id3: 92.50%
---------------------------------
Building for training Split : 8
Accuracy of Id3: 92.59%
---------------------------------
Building for training Split : 9
Accuracy of Id3: 92.67%
---------------------------------
Building for training Split : 0
Accuracy of J48: 100.00%
---------------------------------
Building for training Split : 1
Accuracy of J48: 96.67%
---------------------------------
Building for training Split : 2
Accuracy of J48: 97.78%
---------------------------------
Building for training Split : 3
Accuracy of J48: 98.33%
---------------------------------
Building for training Split : 4
Accuracy of J48: 98.67%
---------------------------------
Building for training Split : 5
Accuracy of J48: 98.89%
---------------------------------
Building for training Split : 6
```

```
Accuracy of J48: 97.14%
---------------------------------
Building for training Split : 7
Accuracy of J48: 97.50%
---------------------------------
Building for training Split : 8
Accuracy of J48: 97.04%
---------------------------------
Building for training Split : 9
Accuracy of J48: 96.67%
---------------------------------
Building for training Split : 0
Accuracy of myID3: 46.67%
---------------------------------
Building for training Split : 1
Accuracy of myID3: 50.00%
---------------------------------
Building for training Split : 2
Accuracy of myID3: 51.11%
---------------------------------
Building for training Split : 3
Accuracy of myID3: 48.33%
---------------------------------
Building for training Split : 4
Accuracy of myID3: 46.67%
---------------------------------
Building for training Split : 5
Accuracy of myID3: 51.11%
---------------------------------
Building for training Split : 6
Accuracy of myID3: 51.43%
---------------------------------
Building for training Split : 7
Accuracy of myID3: 50.00%
---------------------------------
Building for training Split : 8
Accuracy of myID3: 48.15%
---------------------------------
Building for training Split : 9
Accuracy of myID3: 45.33%
---------------------------------
Building for training Split : 0
Accuracy of myC45: 46.67%
---------------------------------
Building for training Split : 1
Accuracy of myC45: 53.33%
---------------------------------
Building for training Split : 2
Accuracy of myC45: 60.00%
---------------------------------
Building for training Split : 3
Accuracy of myC45: 55.00%
---------------------------------
```

```
Building for training Split : 4
Accuracy of myC45: 54.67%
---------------------------------
Building for training Split : 5
Accuracy of myC45: 56.67%
---------------------------------
Building for training Split : 6
Accuracy of myC45: 53.33%
---------------------------------
Building for training Split : 7
Accuracy of myC45: 54.17%
---------------------------------
Building for training Split : 8
Accuracy of myC45: 52.59%
---------------------------------
Building for training Split : 9
Accuracy of myC45: 50.00%
---------------------------------
```

3. **Perbandingan dengan hasil ID3 dan J48**
   Terdapat perbedaan akurasi kumulatif hasil prediksi ID3 dan J48 yang khas untuk dataset yang berbeda. Pada data weather, akurasi kumulasi pada akhir iterasi dari myID3, myC4.5 dan Weka ID3 sama besar. Kemungkinan karena data yang digunakan masih dalam jumlah kecil. Pada dataset iris, perbedaan keempat algoritma semakin jelas. Hal ini disebabkan tidak adanya penanganan multiclass pada algoritma myID3 dan myC4.5.

.

   a. **Data Weather(binary-class)**

Training Split	**Akurasi Kumulatif**			
	Weka ID3	**Weka J48**	**myID3**	**myC4.5**
0	50.00%	50.00%	100.00%	100.00%
1	50.00%	75.00%	75.00%	75.00%
2	66.67%	83.33%	66.67%	66.67%
3	75.00%	87.50%	62.50%	62.50%
4	77.78%	88.89%	66.67%	66.67%
5	80.00%	90.00%	70.00%	70.00%
6	72.73%	81.82%	63.64%	63.64%

7	75.00%	83.33%	66.67%	66.67%
8	76.92%	84.62%	69.23%	69.23%
9	71.43%	85.71%	71.43%	71.43%

b. **Data Iris(multiple-class)**

Training Split	**Akurasi Kumulatif**			
	Weka ID3	**Weka J48**	**myID3**	**myC4.5**
0	93.33%	100.00%	46.67%	46.67%
1	96.67%	96.67%	50.00%	53.33%
2	88.89%	97.78%	51.11%	60.00%
3	90.00%	98.33%	48.33%	55.00%
4	90.67%	98.67%	46.67%	54.67%
5	92.22%	98.89%	51.11%	56.67%
6	92.38%	97.14%	51.43%	53.33%
7	92.50%	97.50%	50.00%	54.17%
8	92.59%	97.04%	48.15%	52.59%
9	92.67%	96.67%	45.33%	50.00%

c. **Klasifikasi unseen data Weather.arff**

Data yang dimasukkan <outlook,temperature,humidity,windy>	**Hasil klasifikasi**			
	Weka ID3	**Weka J48**	**myID3**	**myC4.5**
<sunny, 65, 120, FALSE>	no	no	no	yes
<rainy, 40, 90, TRUE>	?	no	yes	no
<rainy, 80, 40,	no	yes	yes	no

FALSE>				
<overcast, 80, 55, TRUE>	no	no	yes	yes

d. **Klasifikasi unseen data Iris.arff**

Data yang dimasukkan <sepallength,sepalwidth, petallength,petalwidth>	**Hasil klasifikasi**			
	Weka ID3	**Weka J48**	**myID3**	**myC4.5**
<5.2, 4.1, 1.6, 0.2>	Iris-setosa	Iris-setosa	Iris-setosa	Iris-setosa
<7.0, 5.5, 3.1, 1.8>	Iris-versicolor	Iris-versicolor	Iris-virginica	Iris-versicolor
<2.3, 6.9, 7.1, 5.5>	Iris-versicolor	Iris-versicolor	Iris-setosa	Iris-setosa
<5.6, 2.7, 4.2, 1.3>	Iris-setosa	Iris-setosa	Iris-setosa	Iris-setosa

4. **Penanganan permasalahan**
   a. **Penanganan atribut kontinu**
   Atribut kontinu terlebih dahulu diurutkan dari angka terkecil hingga terbesar. Kemudian dibagi setiap turning point. Misalkan terdapat 5 atribut dengan label demikian.
   [<20,yes>, <30,yes>, <40,yes>, <50,no>,<70,yes>]
   Dari data tersebut, Penulis membagi data ke dalam 3 range yaitu <=40, 40<x<70 dan >=70. Setiap data kemudian dimasukkan ke dalam range kategori yang telah dibuat.
   b. **Penanganan missing value**
   Missing value pada atribut segera dibuang ketika ditemukan.

5. **Pembagian Kerja**

NIM	Nama	Hal yang dikerjakan
13512501	Muhammad Nizami	- Implementasi perhitungan entropi myID3 - Implementasi struktur data myID3 - Implementasi splitData - Implementasi pembangunan pohon myID3 - Implementasi serialisasi kelas myID3 (untuk save) myID3
13513039	Ivan Andrianto	- Implementasi perhitungan IG myID3 (menggunakan entropi yang dibuat anggota lain) - Save/Load Model - Numerik to Nominal - Kode Classifier - Input dan test unseen data - 10-fold cross validation
13513071	Wilhelmus Andrian Tanujaya	- CSV to Arff - Filter, remove unused - Resampling - Penanganan atribut kontinu - Input unseen data - MyC4.5