Εξόρυξη Δεδομένων

Στοϊλούδης
Κωνσταντίνος
3212016191

1. Αφαιρέστε την παράμετρο decision (121) και decision_0 (122) από τα δεδομένα σας

Αφού έγινε η ανάγνωση του αρχείου arff σε σειρές και δημιουργήθηκε μια λίστα με attributes αφαίρεσα απο την λίστα τα decision και δεν αναγνώστηκαν τα values τους για αυτο το λόγο

κωδικας:

```
@Slf4j  3 usages   xator
public class ArffReader {
    private File file;  2 usages

    public ArffReader(File file) throws FileNotFoundException {  1 usage   xator
        this.file = file;
    }
    public Dataset loadData() throws IOException {  1 usage   xator
        log.info("reading data ...");
        var bf = new BufferedReader(new FileReader(file));
        var lines = bf.lines().toList();
        var dataset = new Dataset(lines.stream().filter( String s -> s.contains("@relation")).toList().toString());
        var attributes = new ArrayList<String>();
        lines.stream().filter( String line -> line.contains("@attribute")).forEach( String line -> {
              attributes.add(line.split( regex: "@attribute")[1].split( regex: " ")[1].trim());
        });
        var data = lines.stream().filter( String line -> {
            return !line.contains("@attribute") && !line.contains("@relation") && line.contains(",");
        });
        int previous_size = attributes.size();
        data.forEach( String datum -> {
            dataset.load(attributes,datum.split( regex: ","));
        });
//        for(String attribute : attributes)
//            log.info("{}" , attribute);
        attributes.removeIf( String string -> string.contains("decision"));
        log.info("attributes {} reduced to {} by removing decision decision_o",previous_size,attributes.size());
        log.info("data loading complete");
        dataset.setAttributes(attributes);
        return dataset;
    }
}
```

στην εικόνα φαίνεται το parsing του arff αρχείου

επίσης η κλάση dataset που κρατάει τα δεδομένα σε Line αντικείμενα

```java
@Slf4j  54 usages  xator
@Setter
@Getter
public class Dataset {
    private String relation;
    private List<Line> data = new ArrayList<>();
    private List<String> attributes;

    public Dataset(String relation) { this.relation  = relation; }
    public void load(List<String> attributes,String[] values){  1 usage  xator
        try {
            data.add(new Line(attributes,values));
        } catch (Exception e) {
            log.error(e.getMessage());
        }
    }
    public Dataset[] split(int divided){  1 usage  xator
        var datasets = new Dataset[divided];
        int size = this.getData().size();

        for (int i=0; i<divided;i++){
            datasets[i]=new Dataset( relation: "{"+i+"} "+this.relation+" 1/"+divided);
            int newSize = size/divided;
            try {
                datasets[i].getData().addAll(data.subList(i*newSize,i*newSize+newSize));
                datasets[i].setAttributes(this.attributes);
            } catch (Exception e) {
                log.error("{} out of bounds",i);
            }
        }
        for(int i=0; i<divided;i++){
            log.info("{} -> size {}",datasets[i].getRelation(),datasets[i].getData().size());
        }
        return datasets;
    }
```

και η κλάση Line όπου αποθηκεύονται τα δεδομένα σε πίνακα <ιδιότητα,τιμή>

```java
@Slf4j  44 usages  xator
@Setter
@Getter
public static class Line{
    private Map<String,String> line = new HashMap<>();

    @Override  xator
    public String toString() {
        return "Line{" +
                "line=" + line +
                '}';
    }

    public Line(List<String> attributes, String[] values) throws Exception {  1 usage  xator
        var sb = new StringBuilder();
        if (values.length < attributes.size()){
            throw new Exception("values = "+values.length+" but attributes = "+attributes.size());
        }
        for (int i = 0; i < attributes.size(); i++) {
            var attribute = attributes.get(i) ;
            var value =  values[i];
            if(!attribute.equals("decision") && !attribute.equals("decision_o")) {

                line.put(attribute, value.equals("?")?"-1":value);
                log.info("{} -->  {}", attribute, value);
                sb.append(attribute);
                sb.append(" = ");
                sb.append(value);
                if (i != attributes.size() - 1) {
                    sb.append(" , ");
                }
            }
        }
        log.info(sb.toString());
    }
    public String get(String attribute) {  25 usages  xator
        return this.line.get(attribute);
    }
}
```

όπως βλέπετε στην εικόνα οι τιμές αποφεύγουν να αποθηκεύσουν τα decision , decision_o

2. Εφαρμόστε τον αλγόριθμο «Δέντρο απόφασης» και αναφερθείτε αναλυτικά στις παραμέτρους και στην υλοποίηση του αλγορίθμου που χρησιμοποιήσατε. Επιλέξτε τη δημιουργία δέντρου με ψαλίδισμα (pruned). Χρησιμοποιήστε το σύνολο των δεδομένων για εκπαίδευση και αξιολόγηση του δέντρου. Επιλέξτε τις κατάλληλες παραμέτρους όπως εσείς κρίνετε. Τι αποτέλεσμα δίνει ο αλγόριθμος κατά την αξιολόγησή του και ποιο είναι το δέντρο που δημιουργήθηκε;

- Σύντομη παρουσίαση του αλγορίθμου και πλήρης αναφορά στις παραμέτρους και στην υλοποίηση που χρησιμοποιήθηκε
- Πίνακας στον οποίο να φαίνεται η απόδοση ή οι αποδόσεις αν είναι απαραίτητο και σχήμα με το δέντρο που προέκυψε
- Επεξήγηση και αιτιολόγηση της μετρικής που χρησιμοποιήθηκε για την αξιολόγηση του αλγορίθμου

```java
public class TreeMath {  4 usages  xator
    private TreeMath(){}  no usages  xator
    public static Double entropy(double p){  2 usages  xator
        if (p <= 0d) return 0d;
        return -p * Math.log(p) / Math.log(2d);
    }
    public static boolean isNumeric(String number){  no usages  xator
        try {
            Double.parseDouble(number);
            return true;
        }catch (Exception e){
            return false;
        }
    }
}
```

έφτιαξα μία κλάση για την εντροπία και τον έλεγχο των τιμών , αν είναι δηλαδή αριθμός ή όχι.

```java
package com.kstoi.trees;

import com.kstoi.utils.Dataset;

public interface DecTree<Node> {  4 usages  2 implementations  xator
    Node build();  1 usage  2 implementations  xator
    String predict(Node root, Dataset.Line instance);  7 usages  2 implementations  xator
    String toString();  2 implementations  xator
}
```

Το interface για λόγους επαναχρησιμοποίησης κώδικα

```java
@AllArgsConstructor  4 usages  xator
@Slf4j
public class PrunedDecisionTree implements DecTree<com.kstoi.trees.PrunedDecisionTree.Node> {

    private Dataset dataset;
    private int maxDepth;
    private int minSamplesSplit;
    private double minGain;

    @Override  xator
    public String toString() {
        return "PrunedDecisionTree{" +
                "maxDepth=" + maxDepth +
                ", minSamplesSplit=" + minSamplesSplit +
                ", minGain=" + minGain +
                '}';
    }

    private List<String> getColumn(List<Dataset.Line> data, String attribute) {  6 usages  xator
        List<String> column = new ArrayList<>();
        for (Dataset.Line line : data) {
            column.add(line.get(attribute));
        }
        return column;
    }
```

Σε αυτήν την εικόνα φαίνεται η κλάση του pruned decision tree

```java
    private Double entropy(List<String> labels) {  5 usages  xator
        double entropy = 0d;
        Map<String, Integer> labelFrequency = new HashMap<>();
        for (var label : labels) {
            labelFrequency.put(label, labelFrequency.getOrDefault(label,  defaultValue: 0) + 1);
        }
        for (var count : labelFrequency.values()) {
            double probability = (double) count / labels.size();
            entropy += TreeMath.entropy(probability);
        }
        return entropy;
    }

    private Map<String, List<Dataset.Line>> splitDataBasedOnCategory(List<Dataset.Line> data, String attribute) {  2 usages  xator
        var result = new HashMap<String, List<Dataset.Line>>();
        for (var line : data) {
            String val = line.get(attribute);
            result.computeIfAbsent(val,  String k -> new ArrayList<>()).add(line);
        }
        return result;
    }

    private String majorityLabel(List<String> labels) {  3 usages  xator
        Map<String, Integer> counts = new HashMap<>();
        for (String label : labels) {
            counts.put(label, counts.getOrDefault(label,  defaultValue: 0) + 1);
        }
        return Collections.max(counts.entrySet(), Map.Entry.comparingByValue()).getKey();
    }

    private Double informationGain(List<Dataset.Line> data, String attribute, String target) {  1 usage  xator
        double baseEntropy = entropy(getColumn(data, target));
        Map<String, List<Dataset.Line>> subsets = splitDataBasedOnCategory(data, attribute);
        double newEntropy = 0d;
        for (List<Dataset.Line> subset : subsets.values()) {
            double p = (double) subset.size() / data.size();
            newEntropy += p * entropy(getColumn(subset, target));
        }
        return baseEntropy - newEntropy;
    }
```

μεθόδοι για την βασική εντροπία , την πληροφορία που δίνει η καθορισμένη ιδιότητα και η ιδιότητα με τη μεγαλύτερη συχνότητα

```java
private double[] bestThresholdNumeric(List<Dataset.Line> data, String attribute, String target) {  1 usage  xator
    List<Dataset.Line> sorted = new ArrayList<>(data);
    sorted.sort(Comparator.comparing( Line m ->  Double.parseDouble(m.get(attribute))));

    double bestGain = 0d;
    double bestThresh = 0d;
    double baseEntropy = entropy(getColumn(data, target));

    for (int i = 1; i < sorted.size(); i++) {
        double prev = Double.parseDouble(sorted.get(i - 1).get(attribute));
        double curr = Double.parseDouble(sorted.get(i).get(attribute));
        if (prev == curr) continue;

        double thresh = (prev + curr) / 2.0;

        List<Dataset.Line> left = new ArrayList<>();
        List<Dataset.Line> right = new ArrayList<>();

        for (Dataset.Line line : sorted) {
            double val = Double.parseDouble(line.get(attribute));
            if (val <= thresh) left.add(line);
            else right.add(line);
        }

        double newEntropy = (left.size() * entropy(getColumn(left, target))
                + right.size() * entropy(getColumn(right, target))) / data.size();

        double gain = baseEntropy - newEntropy;
        if (gain > bestGain) {
            bestGain = gain;
            bestThresh = thresh;
        }
    }

    return new double[]{bestGain, bestThresh};
}
```

Η bestThresholdNumeric είναι η μέθοδος που χρησιμοποιεί το δέντρο για να αντιμετωπίσει τις αριθμητικές τιμές . Βίσκοντας το βέλτιστο σημείο διαχωρίσμου ώστε να μειωθεί η αβεβαιότητα.

Στην εικόνα βλέπουμε την αρχή της μεθόδου buildTree
αν όλες οι ετικέτες είναι ίδιες
ή αν δεν υπάρχουν άλλες ιδιότητες

ή ο αριθμός των δειγμάτων είναι μικρός
επιστρέφει τον κόμβο
(pruning)

```
for (String attr : attributes) {
    double gain;
    double thresh = 0;
    if (isNumeric.getOrDefault(attr,  defaultValue: false)) {
        double[] res = bestThresholdNumeric(data, attr, targetAttr);
        gain = res[0];
        thresh = res[1];
    } else {
        gain = informationGain(data, attr, targetAttr);
    }

    if (gain > bestGain) {
        bestGain = gain;
        bestAttr = attr;
        if (isNumeric.getOrDefault(attr,  defaultValue: false)) {
            bestThreshold = thresh;
            numeric = true;
        } else {
            bestThreshold = null;
            numeric = false;
        }
    }
}

if (bestGain < minGain || bestAttr == null) {
    node.isLeaf = true;
    node.label = majorityLabel(labels);
    return node;
}

node.attribute = bestAttr;
node.threshold = bestThreshold;

List<String> newAttributes = new ArrayList<>(attributes);
newAttributes.remove(bestAttr);

if (numeric) {
    List<Dataset.Line> left = new ArrayList<>();
    List<Dataset.Line> right = new ArrayList<>();
    for (Dataset.Line row : data) {
        double val = Double.parseDouble(row.get(bestAttr));
        if (val <= bestThreshold) left.add(row);
        else right.add(row);
    }
    node.left = buildTree(left, newAttributes, targetAttr,  depth: depth + 1, isNumeric);
    node.right = buildTree(right, newAttributes, targetAttr,  depth: depth + 1, isNumeric);
} else {
    Map<String, List<Dataset.Line>> subsets = splitDataBasedOnCategory(data, bestAttr);
    for (Map.Entry<String, List<Dataset.Line>> entry : subsets.entrySet()) {
        node.children.put(entry.getKey(),
                buildTree(entry.getValue(), newAttributes, targetAttr,  depth: depth + 1, isNumeric));
    }
}

  log.info("Built node: {}", node);
return node;
```

Η εικόνα του βασικού αλγορίθμου

Σε κάθε κόμβο του δέντρου, ο αλγόριθμος ελέγχει όλα τις διαθέσιμες ιδιότητες και υπολογίζει το information gain που προκύπτει αν γίνει διάσπαση με βάση τη κάθε μια.
Έπειτα, επιλέγει την ιδιότητα με το μεγαλύτερο information gain. Αν αυτό το gain είναι πολύ μικρό (ή δεν υπάρχει κατάλληλο attribute), τότε ο κόμβος μετατρέπεται σε φύλλο και παίρνει ως ετικέτα την πλειοψηφική κλάση.

Εδώ έκανα μια build μέθοδο που ξεχωρίζει τις ιδιότητες σε αυτές που έχουν αριθμητικές τιμές και στις υπόλοιπες.Τέλος καλείται η αναδρομική buildTree και επιστρέφεται ο κόμβος.

```java
public String predict(Node root, Dataset.Line instance) {  7 usages  xator
    if (root.isLeaf) return root.label;

    if (root.threshold != null) {
        double val = Double.parseDouble(instance.get(root.attribute));
        if (val <= root.threshold && root.left != null) {
            return predict(root.left, instance);
        } else if (root.right != null) {
            return predict(root.right, instance);
        } else {
            return root.label;
        }
    }

    String val = instance.get(root.attribute);
    Node child = root.children.get(val);
    if (child != null) {
        return predict(child, instance);
    } else {
        return root.label;
    }
}

@ToString  9 usages  xator
public static class Node {
    String attribute;
    Double threshold;
    String label;
    boolean isLeaf = false;
    Map<String, Node> children = new HashMap<>();
    Node left = null, right = null;
}
```

Εδώ βλέπετε την μέθοδο predict που αναδρομικά διασχίζει το δέντρο και επιστρέφει την πρόβλεψη. Ακόμα , υπάρχει και η κλάση Node που είναι package οι ιδιότητες και αντιπροσοπεύει το δέντρο.

```java
public static <Node> Map<String,Object> testTree(DecTree<Node> tree, Dataset testDataset) {  4 usages  & xator
        var node = tree.build();
        double truePositives=0,trueNegatives=0,falsePositives=0,falseNegatives=0;
        for (Dataset.Line line : testDataset.getData()){
            String prediction = tree.predict(node,line);
            System.out.println(line);
            log.info("line is labeled as {} -> predicted {}",line.get("match}"),prediction);
            if(prediction!=null)
                if(Objects.equals(line.get("match"), prediction) && prediction.equals("1")){
                    truePositives++;
                } else if (Objects.equals(line.get("match"), prediction) && prediction.equals("0")) {
                    trueNegatives++;
                }else if(!Objects.equals(line.get("match"), prediction) && prediction.equals("1")){
                    falsePositives++;
                }else if (!Objects.equals(line.get("match"), prediction) && prediction.equals("1")) {
                    falseNegatives++;
                }
        }
        log.info("TP {} TN {} FP {} FN {}",truePositives,trueNegatives,falsePositives,falseNegatives);
        log.info("precision {}",precision(truePositives,falsePositives));
        log.info("recall {}",recall(truePositives,falseNegatives));
        log.info("specificity {}",specificity(trueNegatives,falsePositives));
        log.info("accuracy {}",accuracy(truePositives,trueNegatives,falsePositives,falseNegatives));
        var result = new HashMap<String,Object>();
        result.put("TP {} TN {} FP {} FN {}", truePositives+" "+trueNegatives+" "+falsePositives+" "+falseNegatives);
        result.put("precision {}",precision(truePositives,falsePositives));
        result.put("recall {}",recall(truePositives,falseNegatives));
        result.put("specificity {}",specificity(trueNegatives,falsePositives));
        result.put("accuracy {}",accuracy(truePositives,trueNegatives,falsePositives,falseNegatives));
        result.put("tree",tree.toString());
        return result;
}
public static double accuracy(double tp,double tn,double fp,double fn) { return (tp+tn)/(tp + tn + fp + fn); }
public static double precision(double tp,double fp) { return tp /(tp+fp); }
public static double recall(double tp,double fn) { return tp /(tp+fn); }
public static double specificity(double tn,double fp) { return tn /(tn+fp); }
```

(στη Main η μέθοδος τεστ και οι μέθοδοι των μετρικών που χρησιμοποιήθηκαν) και στην παρακάτω εικόνα φαίνετε η main με όλα τα τεστ των ερωτημάτων

2.Pruned Decision Tree results
precision -> 0.8818263205013429
recall -> 1.0
TP {} TN {} FP {} FN {} -> 985.0 6866.0 132.0 0.0
tree -> PrunedDecisionTree{maxDepth=4, minSamplesSplit=5, minGain=0.01}
specificity -> 0.9811374678479565
accuracy -> 0.9834648628335212

Precision → Από τα θετικά που βρέθηκαν, πόσα είναι σωστά;
Recall → Από όλα τα πραγματικά θετικά, πόσα βρέθηκαν;
Specificity → Από όλα τα πραγματικά αρνητικά, πόσα αναγνώριστήκαν;
Accuracy → Πόσο συχνά το μοντέλο προβλέπει σωστά συνολικά;

3. Επιλέξτε την δημιουργία δέντρου χωρίς ψαλίδισμα (unpruned). Χρησιμοποιήστε το σύνολο των δεδομένων για εκπαίδευση του δέντρου. Τι αποτέλεσμα δίνει ο αλγόριθμος κατά την αξιολόγησή του και ποιο είναι το δέντρο που δημιουργήθηκε;
- Πίνακας στον οποίο να φαίνεται η απόδοση ή οι αποδόσεις αν είναι απαραίτητο και σχήμα με το δέντρο που προέκυψε
- Επεξήγηση και αιτιολόγηση της μετρικής που χρησιμοποιήθηκε για την αξιολόγηση του αλγορίθμου

3. Decision Tree results
precision -> 1.0
recall -> 1.0
TP {} TN {} FP {} FN {} -> 1380.0 6998.0 0.0 0.0
tree -> DecisionTree{}
pecificity -> 1.0
accuracy -> 1.0

(αφαίρεση των μεταβλητών του prunning όλα τα άλλα ίδια)

4. Παρατηρώντας τα αποτελέσματα, ποια από τις επιλογές (pruned/unpruned) φαίνεται να είναι καλύτερη και γιατί; Τι αναμένουμε να παρατηρήσουμε, ανάλογα με την επιλογή μας, όταν τα δεδομένα δοκιμής είναι άγνωστα;
- επεξήγηση της μετρικής που χρησιμοποιήθηκε για την αξιολόγηση των αλγορίθμων, και των αποτελεσμάτων
- επεξήγηση του τι αναμένουμε να παρατηρήσουμε, ανάλογα με την επιλογή μας, όταν τα δεδομένα δοκιμής είναι άγνωστα;

Unpruned δέντρο:
Τείνει να μαθαίνει τέλεια τα δεδομένα εκπαίδευσης (πολύ υψηλή ακρίβεια στο training).
Όμως, συχνά παθαίνει υπερεκπαίδευση (overfitting): συλλαμβάνει τον θόρυβο και τις ιδιαιτερότητες του training set.
Ως αποτέλεσμα, σε άγνωστα δεδομένα (test set) η απόδοση πέφτει.

Pruned δέντρο:
Είναι πιο “απλό” μοντέλο, με λιγότερους κόμβους.
Μπορεί να χάνει λίγη ακρίβεια στο training set, αλλά γενικεύει καλύτερα.

Άρα, σε νέα/άγνωστα δεδομένα, περιμένουμε καλύτερη απόδοση (λιγότερο overfitting).
Συμπέρασμα: Συνήθως το pruned δέντρο είναι καλύτερο, γιατί ισορροπεί την πολυπλοκότητα με την ικανότητα γενίκευσης.

5. Εφαρμόστε το Δέντρο απόφασης με και χωρίς pruning και χρησιμοποιήστε 50 % των δεδομένων για δεδομένα εκπαίδευσης. Επιλέξτε οπτικοποίηση της καμπύλης precision-recall για κάθε περίπτωση και δώστε λεπτομέρειες για το τι παρατηρείτε. Πότε είναι η απόδοση του αλγορίθμου καλύτερη και γιατί.
- καμπύλες precision-recall και λεπτομέρειες για το τι παρατηρείται
- πίνακας αποτελεσμάτων, επεξήγηση και αναλυτική ερμηνεία των όσων παρατηρούνται

5.Pruned vs NotPruned split 1/2 dataset
PRUNED
precision {} -> 0.29957805907172996
recall {} -> 1.0
TP {} TN {} FP {} FN {} -> 71.0 3355.0 166.0 0.0
tree -> PrunedDecisionTree{maxDepth=4, minSamplesSplit=5, minGain=0.01}
specificity {} -> 0.9528543027548991
accuracy {} -> 0.9537861915367484
REGULAR
Decision Tree results
precision {} -> 0.2805194805194805
recall {} -> 1.0
TP {} TN {} FP {} FN {} -> 108.0 3244.0 277.0 0.0
tree -> DecisionTree{}
specificity {} -> 0.9213291678500426
accuracy {} -> 0.9236704326260677