Tree Classifiers

PU Machine learning mit Java, Weka und UIMA

Nils Reiter,
nils.reiter@uni-koeln.de

December 1, 2020
Winter term 2020/21

Organisatorisches

Übungen

- Studienleistung besteht in der Abgabe *aller** Übungen
- Bitte auch halbfertige Lösungen pushen, zeigen Rede- und Erklärbedarf

*...mit den üblichen Ausnahmen für Krankheit, Coronablues, Liebeskummer etc.

Organisatorisches

- Übungen
  - Studienleistung besteht in der Abgabe *aller** Übungen
  - Bitte auch halbfertige Lösungen pushen, zeigen Rede- und Erklärbedarf
- Anmeldung zu Modulprüfungen: in den nächsten Tagen via Klips

*...mit den üblichen Ausnahmen für Krankheit, Coronablues, Liebeskummer etc.

Organisatorisches

- Übung
  - S
  - B
- Anmel
- Modul
  - N
  - C
  - E
  - M

de

ermin, zu

*...mit den üblichen Ausnahmen für Krankheit, Coronablues, Liebeskummer etc.

Section 1

Exercise 4

```java
public static void main(String[] args) throws Exception {
  // Parse command line options. Not in exercise, but useful
  // This is done with the libarary JewelCLI
  // http://jewelcli.lexicalscope.com
  Options options = CliFactory.parseArguments(Options.class, args);

  // load data set
  File inputFile = new File(options.getInput());
  ArffLoader loader = new ArffLoader();
  loader.setFile(inputFile);
  Instances instances = loader.getDataSet();
  instances.setClassIndex(instances.numAttributes() - 1);

  // Inititalize and use first filter
  StringToNominal filter0 = new StringToNominal();
  filter0.setAttributeRange("first-last");
  filter0.setInputFormat(instances);
  instances = Filter.useFilter(instances, filter0);

  // Inititalize and use second filter
  MergeInfrequentNominalValues filter1 = new MergeInfrequentNominalValues();
  filter1.setAttributeIndices("first-last");
  filter1.setMinimumFrequency(10);
  filter1.setInputFormat(instances);
  instances = Filter.useFilter(instances, filter1);

  // Initialise the classifier, potentially with parameters
  NaiveBayes nb = new NaiveBayes();

  // Evaluation with the built-in handling of cross validation and evaluation
  Evaluation evaluation = new Evaluation(instances);
  evaluation.crossValidateModel(nb, instances, options.getNumberOfFolds(), new Random(options.getRandomSeed()));
  System.out.println(evaluation.toClassDetailsString());
}
```

```
Nov 30, 2020 10:56:39 AM com.github.fommil.jni.JniLoader liberalLoad
INFO: successfully loaded /var/folders/_g/x7_7tgvj3nl3j9mkfgg35t9m0000gn/T/jniloader8373553642229362862netlib-native_sys
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,733    0,122    0,983      0,733    0,840      0,379    0,893     0,986     O
                 0,661    0,051    0,253      0,661    0,366      0,385    0,946     0,515     B-EntityPER
                 0,568    0,086    0,200      0,568    0,296      0,297    0,842     0,360     I-EntityPER
                 0,720    0,053    0,131      0,720    0,222      0,292    0,938     0,256     B-EntityLOC
                 0,700    0,073    0,150      0,700    0,247      0,300    0,893     0,420     I-EntityLOC
                 0,000    0,002    0,000      0,000    0,000      -0,001   0,475     0,000     B-EntityWRK
                 0,000    0,005    0,000      0,000    0,000      -0,002   0,690     0,002     I-EntityWRK
Weighted Avg.    0,723    0,117    0,910      0,723    0,789      0,373    0,892     0,931
```

Potential Issues

- Filters need to know their input format first
  - `filter.setInputFormat(instances)`
- Order of filter application matters
- Do feature removal first, because performance

Potential Issues

- Filters need to know their input format first
  - `filter.setInputFormat(instances)`
- Order of filter application matters
- Do feature removal first, because performance
- Evaluation
  - `weka.classifiers.Evaluation` is easy to use and handles everything
  - Sometimes you need more control (e.g., debugging) → next slide

Reference solution: doCrossValidation(...)

```
65  public static void doCrossValidation(Instances instances, Classifier classifier, Options options) throws Exception {
66
67    // for each fold
68    for (int f = 0; f < options.getNumberOfFolds(); f++) {
69      System.err.print("Fold " + f + ": ");
70
71      // split the data into train and test
72      // (we don't have control over the random generator here)
73      Instances train = instances.trainCV(options.getNumberOfFolds(), f);
74      Instances test = instances.testCV(options.getNumberOfFolds(), f);
75
76      // train classifier on training data
77      classifier.buildClassifier(train);
78
79      // Use Evaluation class to get predictions
80      Evaluation eval = new Evaluation(test);
81      double[] predictions = eval.evaluateModel(classifier, test);
82
83      // print all or the first 100 (whichever is smaller) predictions
84      // from this fold
85      for (int i = 0; i < (Math.min(test.numInstances(), 100)); i++) {
86        System.err.print(test.get(i).classValue() + " ");
87        System.err.print(predictions[i] + "|");
88      }
89      System.err.println();
90    }
91  }
```

Reference solution: doCrossValidation(...)

Output


```
Fold 0: 0.0 0.0|0.0 0.0|0.0 0.0|0.0 4.0|0.0 0.0|0.0 0.0|0.0 0.0|0.0 0.0|0.0 0.0|0.0 0.0|0
Fold 1: 0.0 0.0|0.0 1.0|1.0 2.0|2.0 2.0|0.0 0.0|0.0 0.0|0.0 3.0|0.0 2.0|0.0 0.0|0.0 0.0|0
Fold 2: 0.0 4.0|0.0 3.0|0.0 4.0|0.0 0.0|0.0 3.0|0.0 4.0|0.0 4.0|0.0 0.0|0.0 3.0|0.0 4.0|0
Fold 3: 0.0 0.0|0.0 3.0|0.0 2.0|0.0 2.0|0.0 0.0|0.0 4.0|0.0 3.0|0.0 2.0|0.0 4.0|0.0 2.0|0
Fold 4: 2.0 2.0|0.0 0.0|0.0 0.0|0.0 0.0|0.0 0.0|0.0 2.0|0.0 2.0|0.0 0.0|0.0 0.0|0.0 0.0|0
Fold 5: 0.0 2.0|0.0 3.0|0.0 2.0|0.0 0.0|0.0 0.0|0.0 0.0|1.0 2.0|0.0 4.0|0.0 3.0|0.0 4.0|0
Fold 6: 0.0 0.0|0.0 0.0|0.0 0.0|0.0 0.0|0.0 1.0|0.0 2.0|0.0 0.0|0.0 0.0|0.0 0.0|3.0 3.0|4
Fold 7: 0.0 0.0|0.0 4.0|0.0 0.0|0.0 0.0|0.0 0.0|0.0 0.0|0.0 1.0|0.0 2.0|0.0 4.0|0.0 0.0|3
Fold 8: 0.0 0.0|0.0 0.0|0.0 0.0|1.0 3.0|2.0 2.0|0.0 0.0|0.0 1.0|0.0 2.0|0.0 0.0|3.0 3.0|4
Fold 9: 0.0 1.0|0.0 2.0|0.0 0.0|1.0 1.0|0.0 0.0|0.0 0.0|0.0 0.0|0.0 0.0|0.0 2.0|0.0 0.0|0
```

Random Number Generators

- Computers don't have random numbers (because they are deterministic machines)
- Real randomness can only achieved with specialised hardware

Ran

www.amazon.de/TrueRNG-V3-Hardware-Random-Gen

amazon.de

Computers & Accessories

Hello, Nils
Account & Lists

Returns
& Orders

Discover
Prime

0 Shopping-Basket

Deliver to Nils
Dortmund 44141

Cyber Monday Sale | AmazonBasics | Prime Video | Nils's Amazon

Cyber Monday
Ends in
12 : 16 : 26
HOURS MINUTES SECONDS

Computers | Deals | PCs & Laptops | Tablets | PCs | PC Gaming | Computer Accessories | Computer Components | Monitors | Printers | Best Sellers

Computer & Accessories › Accessories › Cables & Accessories › Cables › USB Cables

Truerng V3 – USB Hardware Zufallszahlengenerator

Brand: ubld.it

4.4 out of 5 stars 25 ratings

Our Price: €97.44 & **FREE Delivery**
Prices for items sold by Amazon include temporarily reduced VAT. Depending on your delivery address, VAT may vary at Checkout. For other items, please see details. Information on the reduced VAT in Germany.

New (4) from €97.44 + FREE Shipping

- High performance Speed: > 400 kilobit/second
- Improved Internal Whitening
- Native support for Windows and Linux

€97.44
& **FREE Delivery**

Arrives: **Dec 18 - 31** Details

Fastest delivery: **Dec 3 - 5** Details

In stock.

Quantity: 1

Add to Basket

Buy Now

Secure transaction

Dispatched from and sold by Boss Bross. For Returns, please check the seller link.

Random Number Generators

- Computers don't have random numbers (because they are deterministic machines)
- Real randomness can only achieved with specialised hardware
- Computers: Pseudo-random numbers
  - Start with a seed value x_0
  - Function to generate x_{n+1} from x_n
    - This is a research area on its own

Random Number Generators

- Computers don't have random numbers (because they are deterministic machines)
- Real randomness can only achieved with specialised hardware
- Computers: Pseudo-random numbers
  - Start with a seed value x_0
  - Function to generate x_{n+1} from x_n
    - This is a research area on its own
- Possible seed value sources
  - Mouse movement (not really random ...)
  - Memory allocation
  - Fixed number (0, 23, 42, ...)
- Why?
  - Predictability of lab conditions (= reproducibility of results)
  - Not used when models are applied productively

Random Number Generators

- Computers don't have random numbers (because they are deterministic machines)
- Real randomness can only achieved with specialised hardware
- Computers: Pseudo-random numbers
  - Start with a seed value x_0
  - Function to generate x_{n+1} from x_n
    - This is a research area on its own
- Possible seed value sources
  - Mouse movement (not really random ...)
  - Memory allocation
  - Fixed number (0, 23, 42, ...)
- Why?
  - Predictability of lab conditions (= reproducibility of results)
  - Not used when models are applied productively

```
1    Random random = new java.util.Random(5);
2    for (int i = 0; i < 10; i++) {
3      System.err.println(random.nextInt());
4    }
```

Random Number Generators

- Computers don't have random numbers (because they are deterministic machines)
- Real randomness can only achieved with specialised hardware
- Computers: Pseudo-random numbers
  - Start with a seed value x_0
  - Function to generate x_{n+1} from x_n
    - This is a research area on its own
- Possible seed value sources
  - Mouse movement (not really random ...)
  - Memory allocation
  - Fixed number (0, 23, 42, ...)
- Why?
  - Predictability of lab conditions (= reproducibility of results)
  - Not used when models are applied productively

```
1    Random random = new java.util.Random(5);
2    for (int i = 0; i < 10; i++) {
3      System.err.println(random.nextInt());
4    }
```

```
-1157408321
 758500184
 379066948
-1667228448
 2099829013
 -236332086
 1983575708
 -745993913
 1926715444
 1836354642
```

Git

- Some files don't belong in the git repository
  - `target/`, because it's compiled code
  - Eclipse meta data files: `.classpath`, `.project`, `.settings`
  - IntelliJ meta data files: `.idea`
  - Netbeans files: ...

Git

- Some files don't belong in the git repository
  - `target/`, because it's compiled code
  - Eclipse meta data files: `.classpath`, `.project`, `.settings`
  - IntelliJ meta data files: `.idea`
  - Netbeans files: ...
- `.gitignore` – A file in a git repository
  - Lists file patterns that are to be ignored by git
  - List of patterns for various file types: `https://github.com/github/gitignore`

Section 2

Decision Trees

Recap: The Machine-Learning Recipe

1. Your problem
   - What are its structural properties?
   - What do you believe/know is relevant information to make a prediction?
   - To what extent can humans solve it?
   - Do you have non-formalisable requirements?

Recap: The Machine-Learning Recipe

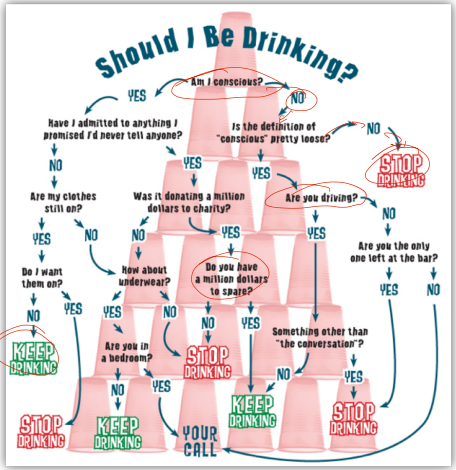
1. Your problem
   - What are its structural properties?
   - What do you believe/know is relevant information to make a prediction?
   - To what extent can humans solve it?
   - Do you have non-formalisable requirements?
2. Pick an appropriate algorithm
   - There are many out there
   - Appropriate: It should fit to your problem!
   - Gather training/testing data

Recap: The Machine-Learning Recipe

1. Your problem
   - What are its structural properties?
   - What do you believe/know is relevant information to make a prediction?
   - To what extent can humans solve it?
   - Do you have non-formalisable requirements?
2. Pick an appropriate algorithm
   - There are many out there
   - Appropriate: It should fit to your problem!
   - Gather training/testing data
3. Evaluate fairly
   - What's the performance of a realistic and optimised baseline?
   - What's an appropriate evaluation metric?
   - Report evaluation results including all settings and constraints

Decision Trees

Prediction Model – Toy Example



- What are the instances?
  - Situations we are in
    (this is not really automatisable)
- What are the features?
  - Consciousness
  - Clothing situation
  - Promises made
  - Whether we are driving
  - ...

Decision Trees

Prediction Model

- Each non-leaf node in the tree represents one feature
- Each leaf node represents a class label
- Each branch at this node represents one possible feature value
  - Number of branches = number of possible values for this feature

Decision Trees

Prediction Model

- Each non-leaf node in the tree represents one feature
- Each leaf node represents a class label
- Each branch at this node represents one possible feature value
  - Number of branches = number of possible values for this feature
- Make a prediction for x:
  1. Start at root node
  2. If it's a leaf node
     - assign the class label
  3. Else
     - Check node which feature is to be tested
     - Extract feature value
     - Follow corresponding branch
     - Go to 2

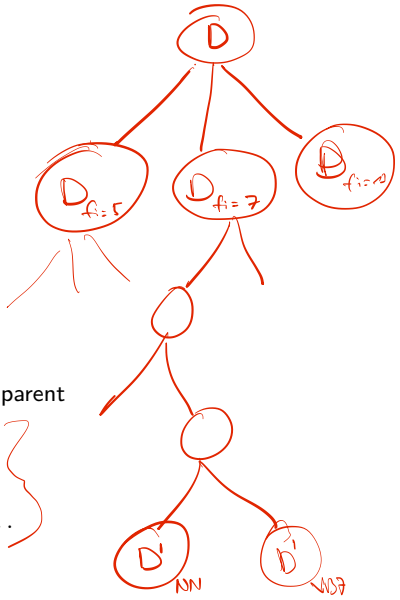
Weka

- `weka.classifiers.trees.J48`
  - "Class for generating a pruned or unpruned C4.5 decision tree" Quinlan (1993)
- C4.5: Extension of ID3 algorithm Quinlan (1986)
- `weka.classifiers.trees.RandomForest` Breiman (2001)

ID3

Quinlan (1986)

- Core idea: The tree represents splits of the training data
  1. Start with the full data set D_{train} as D
  2. If D only contains members of a single class:
     - Done, the node is labeled with the class
  3. Else if there are no more features to select:
     - Done, the node is labeled with the majority class
  4. Else if D is empty
     - Done, the node is labeled with the majority class of its parent
  5. Else:
     - Select a feature f_i
     - Extract feature values of all instances in D
     - Split the data set according to f_i: $D = D_v \cup D_w \cup D_u \ldots$
     - For each subset as D, go back to 2
- Remaining question: How to select features?

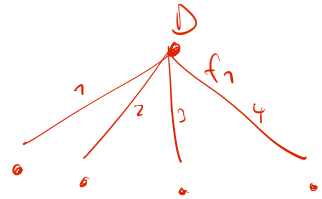


Feature Selection

- What is a good feature?
  - One that maximizes homogeneity in the split data set

Feature Selection

- What is a good feature?
  - One that maximizes homogeneity in the split data set
- "Homogeneity"
  - Increase
    $\{\spadesuit\spadesuit\spadesuit\heartsuit\} = \{\heartsuit\} \cup \{\spadesuit\spadesuit\spadesuit\}$
  - No increase
    $\{\spadesuit\spadesuit\spadesuit\heartsuit\} = \{\spadesuit\} \cup \{\spadesuit\spadesuit\heartsuit\}$



Feature Selection

- What is a good feature?
  - One that maximizes homogeneity in the split data set
- "Homogeneity"
  - Increase
    $\{♠♠♠♡\} = \{♡\} \cup \{♠♠♠\}$ ← better split!
  - No increase
    $\{♠♠♠♡\} = \{♠\} \cup \{♠♠♡\}$
- Homogeneity: Entropy/information Shannon (1948)

Feature Selection

- What is a good feature?
  - One that maximizes homogeneity in the split data set
- "Homogeneity"
  - Increase
    $\{♠♠♠♡\} = \{♡\} \cup \{♠♠♠\}$ ← better split!
  - No increase
    $\{♠♠♠♡\} = \{♠\} \cup \{♠♠♡\}$
- Homogeneity: Entropy/information Shannon (1948)
- Rule: Always select the feature with the highest *information gain* (IG)
  - (= the highest reduction in entropy = the highest increase in homogeneity)

Feature Selection

Entropy (Shannon, 1948)

number of classes present in X

relative frequency of the class

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

Examples (with $b = 2$)

- $H(\{\spadesuit\spadesuit\spadesuit\spadesuit\}) = -\frac{4}{4}\log_2\frac{4}{4} = 0$
- $H(\{\spadesuit\spadesuit\spadesuit\heartsuit\}) = -\left(\underbrace{\frac{3}{4}\log_2\frac{3}{4}}_{\spadesuit} + \underbrace{\frac{1}{4}\log_2\frac{1}{4}}_{\heartsuit}\right) = 0.562$
- $H(\{\spadesuit\spadesuit\heartsuit\heartsuit\}) = \ldots = 0.693 = H(\{\spadesuit\spadesuit\spadesuit\heartsuit\heartsuit\heartsuit\})$ $\quad H(\{\spadesuit\clubsuit\heartsuit\diamondsuit\}) = 1.386294$

Feature Selection (2)

Example

$$
\begin{aligned}
H(\{\spadesuit\spadesuit\spadesuit\heartsuit\}) &= H([3,1]) = 0.562 \\
H(\{\heartsuit\}) &= H([1]) = 0 \\
H(\{\spadesuit\spadesuit\spadesuit\}) &= H([3]) = 0
\end{aligned}
$$

$$
\begin{aligned}
H(\{\spadesuit\spadesuit\spadesuit\heartsuit\}) &= H([3,1]) = 0.562 \\
H(\{\spadesuit\}) &= H([1]) = 0 \\
H(\{\spadesuit\spadesuit\heartsuit\}) &= H([2,1]) = 0.637
\end{aligned}
$$

Feature Selection (3)

Example

$$H(\{\spadesuit\spadesuit\spadesuit\heartsuit\}) = 0.562 \qquad H(\{\spadesuit\spadesuit\spadesuit\heartsuit\}) = 0.562$$
$$H(\{\heartsuit\}) = 0 \qquad H(\{\spadesuit\}) = 0$$
$$H(\{\spadesuit\spadesuit\spadesuit\}) = 0 \qquad H(\{\spadesuit\spadesuit\heartsuit\}) = 0.637$$

$$IG(f_1) = H(\{\spadesuit\spadesuit\spadesuit\heartsuit\}) - \varnothing\left(H(\{\heartsuit\}), H(\{\spadesuit\spadesuit\spadesuit\})\right) = 0.562 - 0 = 0.562$$

$$IG(f_2) = H(\{\spadesuit\spadesuit\spadesuit\heartsuit\}) - \varnothing\left(H(\{\spadesuit\}), H(\{\spadesuit\spadesuit\heartsuit\})\right)$$
$$= 0.562 - \left(\frac{3}{4}0.637 + \frac{1}{4}0\right) = 0.562 - 0.477 = 0.085$$

C4.5

Quinlan (1993)

ID3, but with modifications:

- Continuous attributes
- Data with missing values
- Pruning the tree

Continuous Attributes

The training cases T are first sorted on the values of the attribute A being considered. There are only a finite number of these values, so let us denote them in order as $\{v_1, v_2, \ldots v_m\}$. Any threshold value lying between v_i and v_{i+1} will have the same effect of dividing the cases into those whose value of the attribute A lies in $\{v_1, v_2, \ldots, v_i\}$ and those whose value is in $\{v_{i+1}, v_{i+2}, \ldots, v_m\}$. There are thus only $m - 1$ possible splits on A, all of which are examined. *(Quinlan, 1993, p. 25)*

Continuous Attributes

> *The training cases T are first sorted on the values of the attribute A being considered. There are only a finite number of these values, so let us denote them in order as $\{v_i, v_2, \ldots v_m\}$. Any threshold value lying between v_i and v_{i+1} will have the same effect of dividing the cases into those whose value of the attribute A lies in $\{v_1, v_2, \ldots, v_i\}$ and those whose value is in $\{v_{i+1}, v_{i+2}, \ldots, v_m\}$. There are thus only $m-1$ possible splits on A, all of which are examined.* (Quinlan, 1993, p. 25)

I.e., we use all possible threshold values to split continuous features in two sub sets, and select the one with the highest information gain as usual.

Missing Values

Quinlan (1993, pp. 28 f.)

- Calculate information gain as before, but
  - Take into account only instances whose attribute values are known
  - Weight information gain by the portion of known values

Tree Pruning

> *In my experience, almost all decision trees can benefit from simplification. (Quinlan, 1993, p. 36)*

- Decision trees tend to overfit (e.g., with an 'id feature')
- Pruning the tree prevents that
- C4.5: Post-training pruning
  - Instead of stopping to expand the tree during training

Tree Pruning

In my experience, almost all decision trees can benefit from simplification. (Quinlan, 1993, p. 36)

- Decision trees tend to overfit (e.g., with an 'id feature')
- Pruning the tree prevents that
- C4.5: Post-training pruning
  - Instead of stopping to expand the tree during training
- General idea:
  - For each non-leaf node: Check if pruning reduces error rate
  - Error rate can't accuracy measured on training data, because all tree changes decrease accuracy

Tree Pruning

Pessimistic Pruning

- Pruning decision based on confidence intervals
  - For each node, we can calculate the observed error rate f
  - We are interested in the real error rate p
  - Given a level of confidence, calculate the possible worst case (= upper bound) error rate
  - If the pruned tree has better worst case error rate, prune it

Tree Pruning

Pessimistic Pruning

- Pruning decision based on confidence intervals
  - For each node, we can calculate the observed error rate f
  - We are interested in the real error rate p
  - Given a level of confidence, calculate the possible worst case ($=$ upper bound) error rate
  - If the pruned tree has better worst case error rate, prune it
- Two pruning operations. Selection based on conf. intervals
  - Replace subtree with leaf
  - Replace subtree with one of its branches

Tree Pruning

Pessimistic Pruning

- Pruning decision based on confidence intervals
  - For each node, we can calculate the observed error rate f
  - We are interested in the real error rate p
  - Given a level of confidence, calculate the possible worst case ($=$ upper bound) error rate
  - If the pruned tree has better worst case error rate, prune it
- Two pruning operations. Selection based on conf. intervals
  - Replace subtree with leaf
  - Replace subtree with one of its branches

Reduced Error Pruning

- An alternative pruning strategy, not used by C4.5
- Prune by estimated errors on held-out data set

J48

Options

- C4.5 pruning options
  - `-C <confidence level>`: Pruning confidence level default 0.25

J48

Options

- C4.5 pruning options
  - `-C <confidence level>`: Pruning confidence level
- Reduced error pruning
  - `-R`: Reduced error pruning
  - `-N <number of folds>`

J48

Options

- C4.5 pruning options
  - `-C <confidence level>`: Pruning confidence level
- Reduced error pruning
  - `-R`: Reduced error pruning
  - `-N <number of folds>`
- Other pruning options
  - `-U`: Turn (all) pruning off
  - `-S`: When pruning, don't replace subtrees with branches, only with leafs
  - `-M <number of instances>`: pre-pruning. Only split it at least `<number of instances>` remain in a subset

J48

Options

- C4.5 pruning options
  - `-C <confidence level>`: Pruning confidence level
- Reduced error pruning
  - `-R`: Reduced error pruning
  - `-N <number of folds>`
- Other pruning options
  - `-U`: Turn (all) pruning off
  - `-S`: When pruning, don't replace subtrees with branches, only with leafs
  - `-M <number of instances>`: pre-pruning. Only split it at least `<number of instances>` remain in a subset
- Other options
  - `-B`: Only binary splits
  - `-doNotMakeSplitPointActualValue`: Do not enforce split points of numeric features to be actual values

Source: `https://weka.8497.n7.nabble.com/Details-of-J48-pruning-parameters-td42456.html`

J48 Pruning Options

Example

- Data set: German credit
- Setup: 10-fold cross validation

J48 Pruning Options

Example

- Data set: German credit
- Setup: 10-fold cross validation

Setting	Weighted F1	Tree depth
No pruning	0.718	10
C4.5 pruning	0.751	7
Reduced error pruning	0.719	8

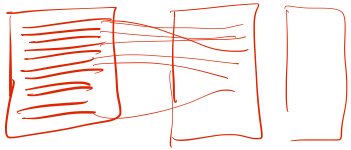
Random Forest

Breiman (2001)

- A tree-based classifier that performs well on many tasks
  - Well suited for limited/small data set sizes

Random Forest

Breiman (2001)



- A tree-based classifier that performs well on many tasks
  - Well suited for limited/small data set sizes
- Bagging
  - Generate variants of the training set by sampling *with replacement* from it
  - Train a decision tree on each variant
  - Create a prediction with all trees, and return the average / majority

Random Forest

Breiman (2001)

- A tree-based classifier that performs well on many tasks
  - Well suited for limited/small data set sizes
- Bagging
  - Generate variants of the training set by sampling *with replacement* from it
  - Train a decision tree on each variant
  - Create a prediction with all trees, and return the average / majority
- Feature bagging
  - At each split, choose between a random subset of features
  - Avoids very similar trees, if one feature is a strong predictor
- Weka
  - `weka.classifiers.trees.RandomForest`

Decision Trees

Summary

- Classification algorithm
- Built around trees, recursive learning and prediction
- Pros
  - Highly transparent
  - Reasonably fast
  - Dependencies between features can be incorporated into the model
- Cons
  - No pairwise dependencies
  - May lead to overfitting – but pruning and bagging help
- Variants exist (e.g., CART)
- Random forest

Exercise 5