# Data Mining III – Lesson 3

Tamara B. Sipes, Ph.D.

# Last Time

- Mining of the CARS dataset, beginning to end:
  - **Problem definition**
  - **Data Preparation**
  - Data Mining
  - Evaluation
  - Presentation

- The focus was on problem definition and data preparation

# Lesson 3 Overview

- Mining of the Image Segmentation dataset, beginning to end:
  - Problem definition
  - Data Preparation
  - **Data Mining**
  - **Evaluation**
  - Presentation

# Instructions

- Hands-on lesson #2
- Again, please have both the Lesson 3 window opened and the weka Explorer window as well
- Follow the step by step instructions, and perform the modeling of the dataset yourself as well

- Let's get started!

# Dataset

- ImageSegmentationData.arff can be found under the Class Resources
- TO DO:
  - Open weka 3.5.7 or similar
  - Choose Explorer from the Applications menu
  - Read in ImageSegmentationData.arff

# Dataset Description

- Image Segmentation data, provided by the Vision Group at University of Massachusetts
- The instances were drawn randomly from a database of 7 outdoor images
- The images were segmented by hand to create a classification for every pixel
- Each instance in the data is a 3x3 region
- Number of Instances: 1500
- Number of Attributes: 20 attributes

# Attribute Information

- 1. region-centroid-col: the column of the center pixel of the region
- 2. region-centroid-row: the row of the center pixel of the region.
- 3. region-pixel-count: the number of pixels in a region = 9
- 4. short-line-density-5: the results of a line extraction algorithm that counts how many lines of length 5 (any orientation) with low contrast, less than or equal to 5, go through the region
- 5. short-line-density-2: same as short-line-density-5 but counts lines of high contrast, greater than 5
- 6. vedge-mean: measure the contrast of horizontally adjacent pixels in the region (used as a vertical edge detector)

# Attribute Information

- 7. vegde-sd: (see 6)
- 8. hedge-mean: measures the contrast of vertically adjacent pixels. Used for horizontal line detection
- 9. hedge-sd: (see 8)
- 10. intensity-mean:

  the average over the region of (R + G + B)/3
- 11. rawred-mean: the average over the region of the R value
- 12. rawblue-mean: the average over the region of the B value
- 13. rawgreen-mean: the average over the region of the G value

# Attribute Information

- 14. exred-mean:

  measure the excess red: (2R - (G + B))
- 15. exblue-mean:

  measure the excess blue: (2B - (G + R))
- 16. exgreen-mean:

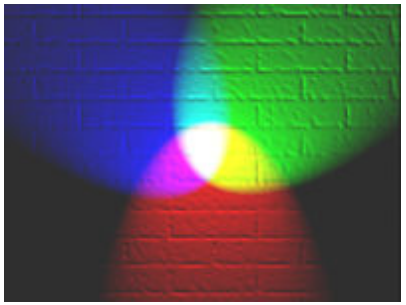
  measure the excess green: (2G - (R + B))
- 17. value-mean: 3-d nonlinear transformation of RGB (using the Foley and VanDam algorithm, in of Interactive Computer Graphics)
- 18. saturatoin-mean: (see 17)
- 19. hue-mean: (see 17)

# Class Variable

- IMAGE – the image classification
- Nominal variable
- Class variable distribution:
  - Brick face
  - Sky
  - Foliage
  - Cement
  - Window
  - Path
  - grass

# RGB Information

- The **RGB color model** is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors
- The name of the model comes from the initials of the three additive primary colors, red, green, and blue



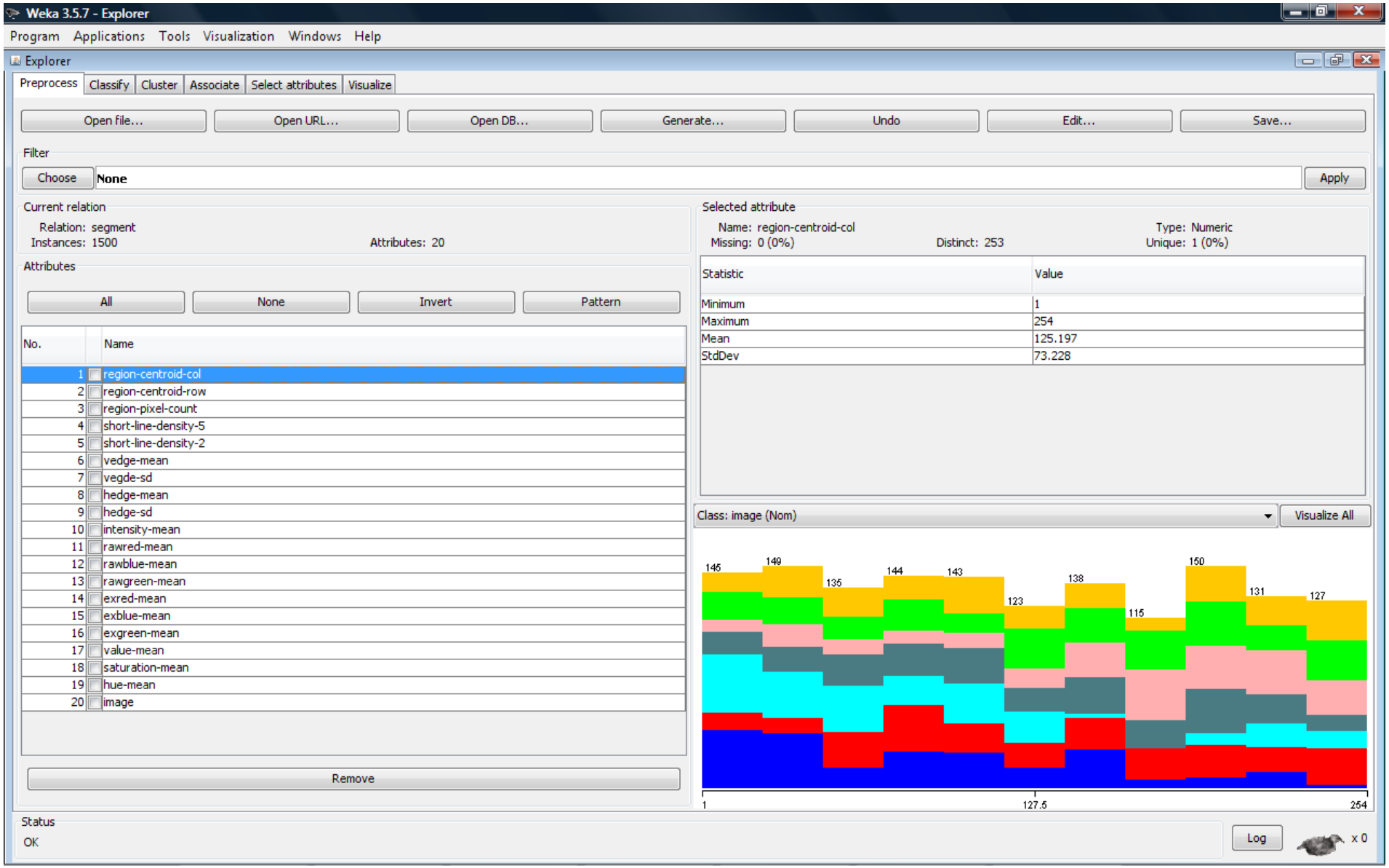
# The Beginning: Problem Definition

- First things first:
  - What is the problem definition?
  - How do we find out?
  - What are we trying to accomplish?
  - Let's find out

# Problem Definition

- Talking to the problem owner, we find out:
  - A predictive classification model is needed to predict an image from a 3x3 pixel segment of an image.
  - The research group is studying human vision, and this model would help increase their understanding of how human vision functions.
  - In other words, we need to predict the IMAGE variable

# Let's Look at the Data!

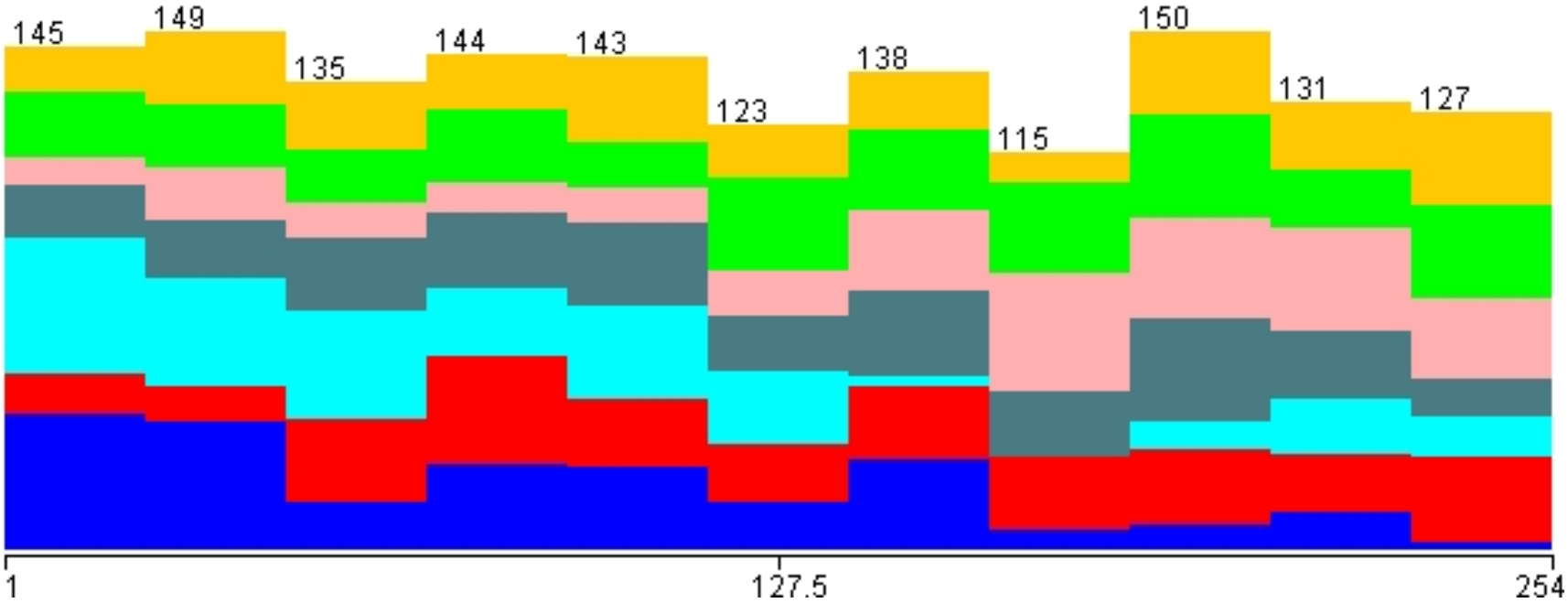


# Attributes

- Number of attributes:  20 attributes
  - 19 numeric attributes
  - 1 nominal attribute

- Let's examine all the attributes in the data

# region-centroid-col



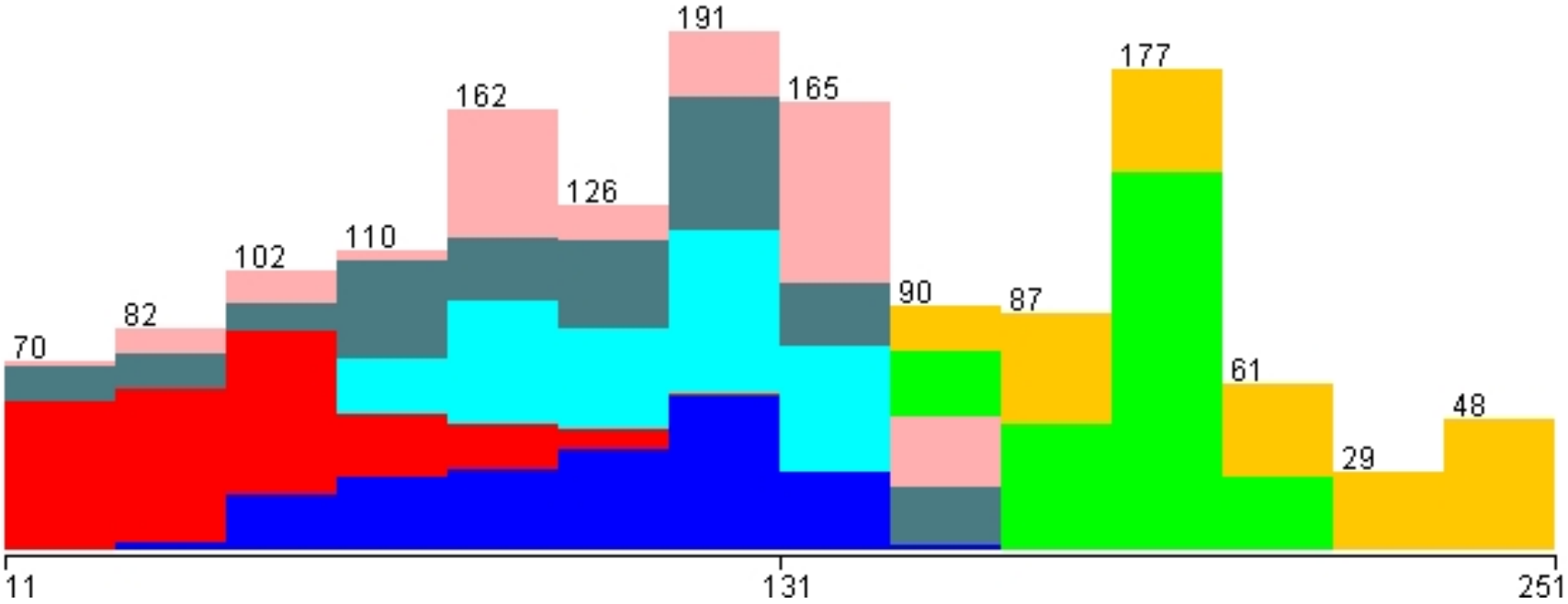
# region-centroid-col: Details

Selected attribute

Name: region-centroid-col Type: Numeric
Missing: 0 (0%) Distinct: 253 Unique: 1 (0%)

| Statistic | Value |
|---|---|
| Minimum | 1 |
| Maximum | 254 |
| Mean | 125.197 |
| StdDev | 73.228 |

# region-centroid-row



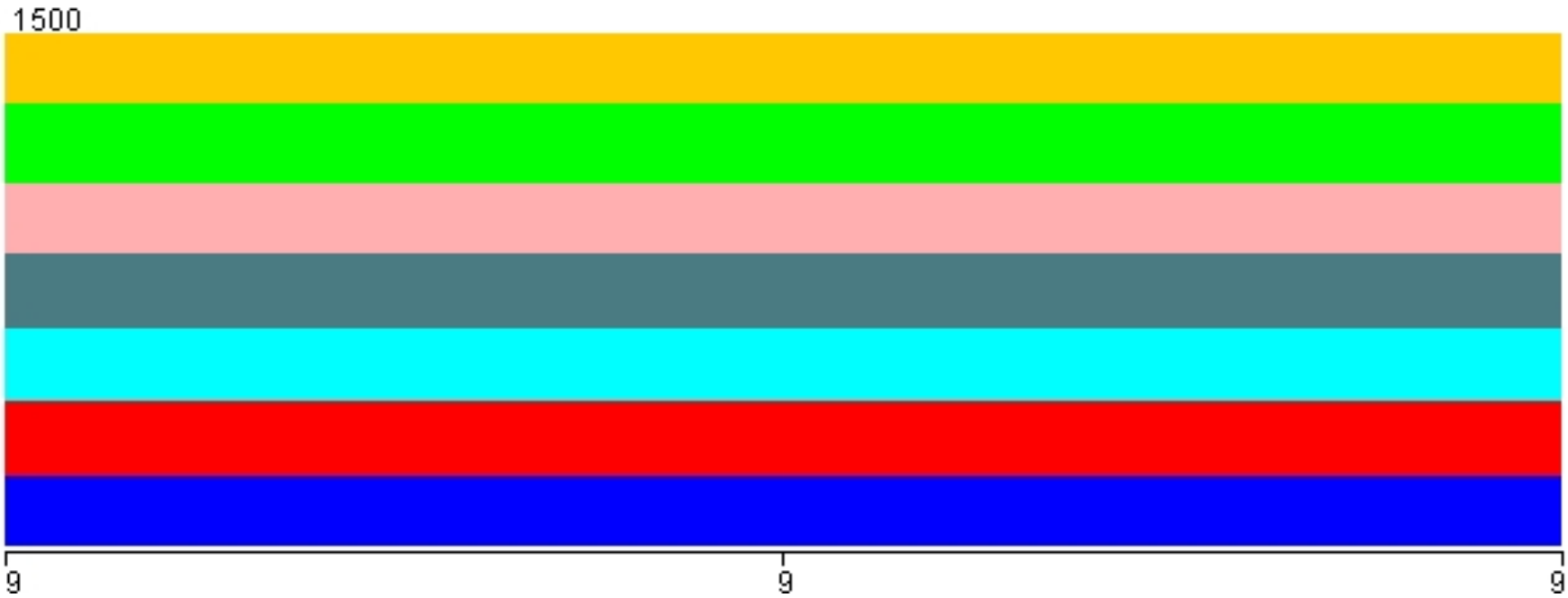
# region-centroid-row: Details

Selected attribute

Name: region-centroid-row
Missing: 0 (0%)
Distinct: 235
Type: Numeric
Unique: 16 (1%)

| Statistic | Value |
|---|---|
| Minimum | 11 |
| Maximum | 251 |
| Mean | 123.76 |
| StdDev | 57.73 |

# region-pixel-count



# region-pixel-count: Details

Selected attribute

Name: region-pixel-count  Type: Numeric
Missing: 0 (0%)  Distinct: 1  Unique: 0 (0%)

| Statistic | Value |
|---|---|
| Minimum | 9 |
| Maximum | 9 |
| Mean | 9 |
| StdDev | 0 |

# Feedback

- Remove the region-pixel-count variable as it is a constant variable
- Select region-pixel-count (box to the left)
- Click on Remove button underneath the variable list
- Check to make sure there are 19 attributes total

# Continue examining the attributes

- Now, on your own, examine the rest of the attributes
- What do you think?
- Anything to discard? Modify?
- Any missing values?

- We will keep all the rest of the attributes

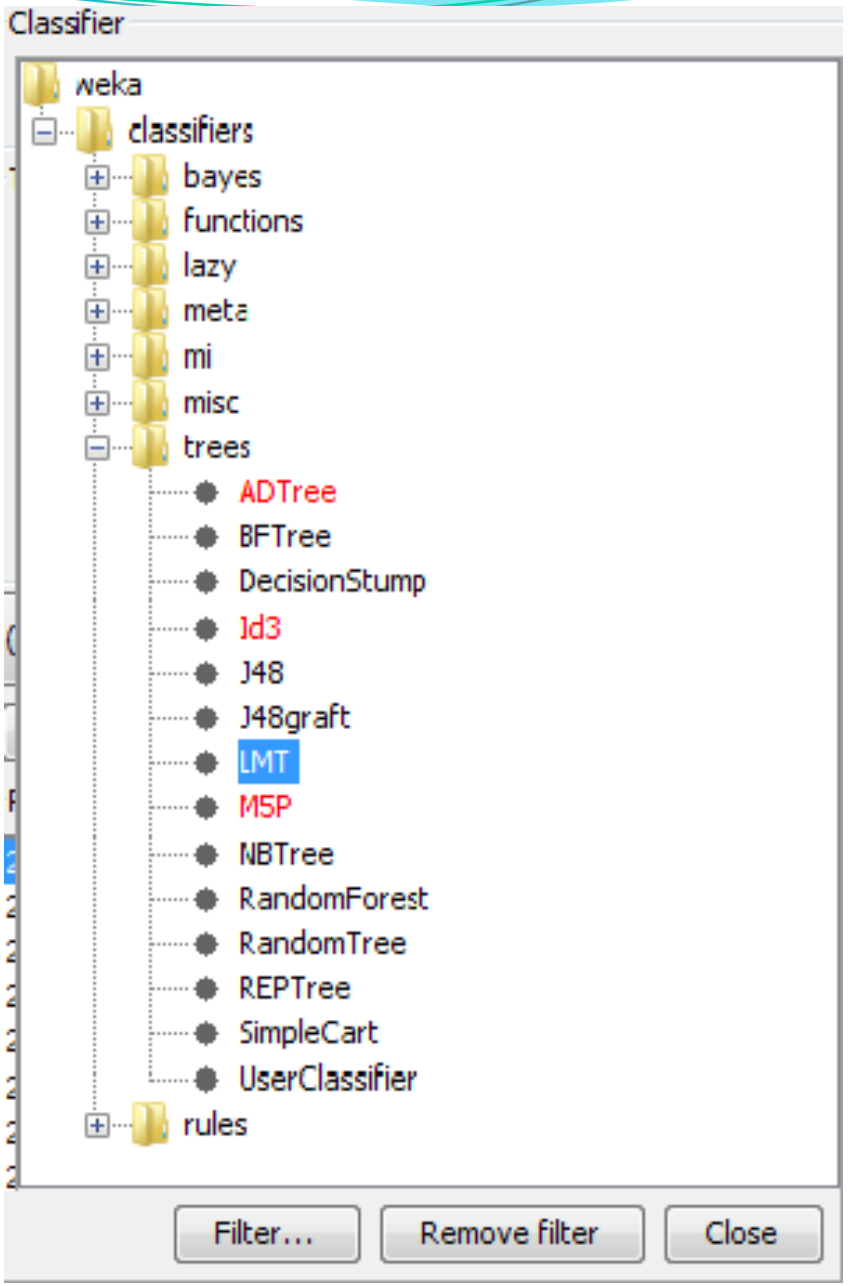
# Now, Let's Start Mining the Data

- We are predicting IMAGE
- Nominal class variable
- Candidate methods:
  - C4.5 (J48)
  - RepTree
  - Decision List
  - ANN
  - SVM
  - other

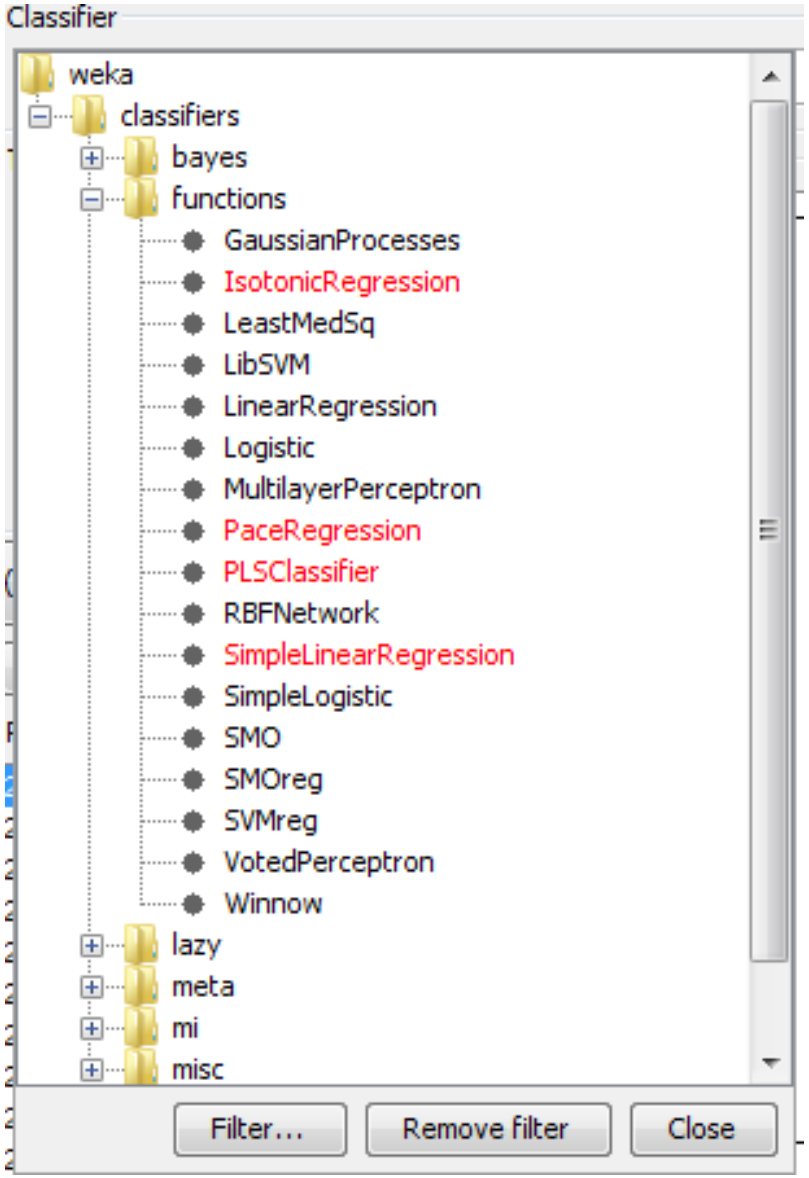
# Selecting the Appropriate Method

- Classify tab
- Click on “Classify” button
- A tree hierarchy of methods
- Which one to use?
- Let’s examine them

# Tree Methods



# Function Methods



# Filtering Methods

- Click on filter and choose “nominal”
- Color tags:
  - Red – does not support the type(s) of attributes selected
  - Blue – most likely does support

# Other Methods

- Bayes
- Lazy
- Meta
- Mi (multi-instance based learners)
- Misc
- Rules (PART, C5 rules, etc.)

# First Iteration

- Let's run the RepTree method on our dataset
- Default parameters
- weka.classifiers.trees.REPTree -M 2 -V 0.0010 -N 3 -S 1 -L -1
- 10-fold cross-validation

# First Iteration: Model

- Size of the tree : 37

```
REPTree
============
region-centroid-row < 155.5
|   rawred-mean < 27.28
|   |   hue-mean < -1.89
|   |   |   hue-mean < -2.21
|   |   |   |   region-centroid-row < 146.5 : foliage (72/1) [38/1]
|   |   |   |   region-centroid-row >= 146.5 : cement (2/0) [2/1]
|   |   |   hue-mean >= -2.21
|   |   |   |   rawred-mean < 2.61
|   |   |   |   |   hue-mean < -2.09
...
```

# First Iteration: Results

| | | |
|---|---|---|
| Correctly Classified Instances | 1414 | 94.2667 % |
| Incorrectly Classified Instances | 86 | 5.7333 % |
| Kappa statistic | 0.9331 | |
| Mean absolute error | 0.0213 | |
| Root mean squared error | 0.1212 | |
| Relative absolute error | 8.6964 % | |
| Root relative squared error | 34.6366 % | |
| Total Number of Instances | 1500 | |

# First Iteration: Results

=== Detailed Accuracy By Class ===

| TP Rate | FP Rate | Precision | Recall | F-Measure | ROC Area | Class |
|---|---|---|---|---|---|---|
| 0.951 | 0.004 | 0.975 | 0.951 | 0.963 | 0.987 | brickface |
| 1 | 0 | 1 | 1 | 1 | 1 | sky |
| 0.894 | 0.019 | 0.882 | 0.894 | 0.888 | 0.962 | foliage |
| 0.905 | 0.006 | 0.961 | 0.905 | 0.932 | 0.979 | cement |
| 0.873 | 0.028 | 0.832 | 0.873 | 0.852 | 0.967 | window |
| 0.987 | 0.006 | 0.967 | 0.987 | 0.977 | 0.994 | path |
| 0.981 | 0.003 | 0.981 | 0.981 | 0.981 | 0.992 | grass |

# First Iteration: Results

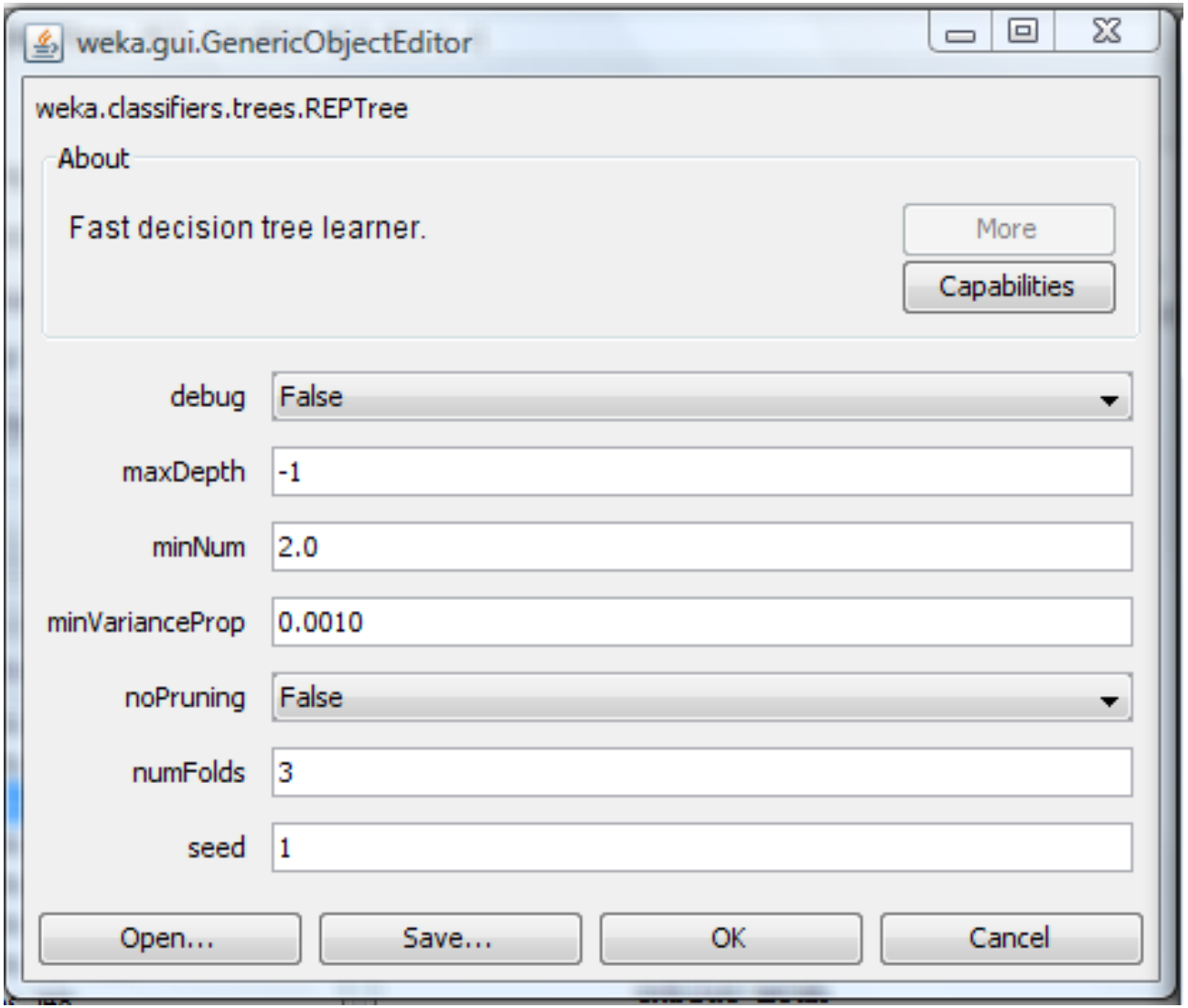
=== Confusion Matrix ===

| a | b | c | d | e | f | g | <-- classified as |
|---|---|---|---|---|---|---|---|
| 195 | 0 | 3 | 0 | 7 | 0 | 0 | a = brickface |
| 0 | 220 | 0 | 0 | 0 | 0 | 0 | b = sky |
| 1 | 0 | 186 | 3 | 18 | 0 | 0 | c = foliage |
| 0 | 0 | 3 | 199 | 9 | 7 | 2 | d = cement |
| 3 | 0 | 19 | 4 | 178 | 0 | 0 | e = window |
| 0 | 0 | 0 | 1 | 0 | 233 | 2 | f = path |
| 1 | 0 | 0 | 0 | 2 | 1 | 203 | g = grass |

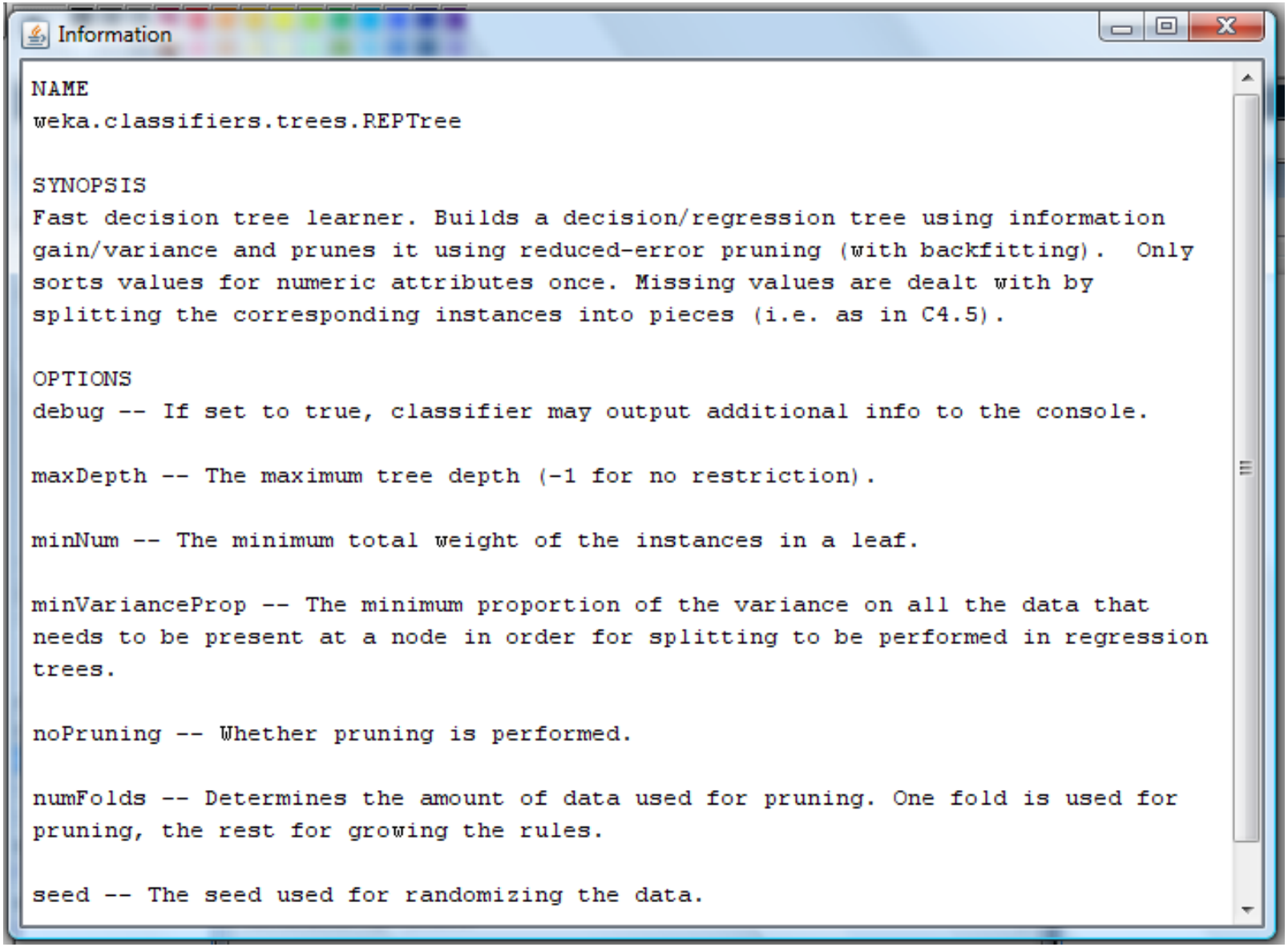
# Next Iteration

- What's next?
- Other methods of interest?
- Two options:
  - 1 – Modify the current method's parameters
  - 2 – Try another method with default parameters
- Let's perform option #1

# Iteration #2


weka.gui.GenericObjectEditor
weka.classifiers.trees.REPTree
About
Fast decision tree learner.
More
Capabilities
debug
False
maxDepth
-1
minNum
2.0
minVarianceProp
0.0010
noPruning
False
numFolds
3
seed
1
Open...
Save...
OK
Cancel

# Iteration #2


Information
NAME
weka.classifiers.trees.REPTree
SYNOPSIS
Fast decision tree learner. Builds a decision/regression tree using information gain/variance and prunes it using reduced-error pruning (with backfitting). Only sorts values for numeric attributes once. Missing values are dealt with by splitting the corresponding instances into pieces (i.e. as in C4.5).
OPTIONS
debug -- If set to true, classifier may output additional info to the console.
maxDepth -- The maximum tree depth (-1 for no restriction).
minNum -- The minimum total weight of the instances in a leaf.
minVarianceProp -- The minimum proportion of the variance on all the data that needs to be present at a node in order for splitting to be performed in regression trees.
noPruning -- Whether pruning is performed.
numFolds -- Determines the amount of data used for pruning. One fold is used for pruning, the rest for growing the rules.
seed -- The seed used for randomizing the data.

# Iteration #2

- Change the minNum (the minimum number of instances in a leaf) to 4:
- weka.classifiers.trees.REPTree -M 4 -V 0.0010 -N 3 -S 1 -L -1
- Correctly Classified Instances 1417 94.4667 %
- Incorrectly Classified Instances 83 5.5333 %

# Iteration #2

- Change the minNum (the minimum  number of instances in a leaf) back to 2
- Change the minVarianceProp to: 0.0050
- Results:  94.2667 %

# Iteration #2

- Change the minVarianceProp back to: 0.0010
- Change the seed to: 333
- Results: 94.4667 %

# Iteration #3

- Change the method!
- Let's try the C4.5
- J48, with default parameters

- weka.classifiers.trees.J48 -C 0.25 -M 2

- Resulting tree:
  - Number of Leaves : 34
  - Size of the tree : 67

# Iteration #3: Results

| | | |
|---|---|---|
| Correctly Classified Instances | 1436 | 95.7333 % |
| Incorrectly Classified Instances | 64 | 4.2667 % |
| Kappa statistic | 0.9502 | |
| Mean absolute error | 0.0138 | |
| Root mean squared error | 0.1057 | |
| Relative absolute error | 5.6471 % | |
| Root relative squared error | 30.2115 % | |
| Total Number of Instances | 1500 | |

# Iteration #3: Results

=== Detailed Accuracy By Class ===

| TP Rate | FP Rate | Precision | Recall | F-Measure | ROC Area | Class |
|---|---|---|---|---|---|---|
| 0.956 | 0.004 | 0.975 | 0.956 | 0.966 | 0.981 | brickface |
| 1 | 0.001 | 0.995 | 1 | 0.998 | 1 | sky |
| 0.942 | 0.018 | 0.895 | 0.942 | 0.918 | 0.975 | foliage |
| 0.941 | 0.009 | 0.945 | 0.941 | 0.943 | 0.978 | cement |
| 0.877 | 0.017 | 0.891 | 0.877 | 0.884 | 0.961 | window |
| 0.987 | 0.001 | 0.996 | 0.987 | 0.991 | 0.997 | path |
| 0.99 | 0 | 1 | 0.99 | 0.995 | 1 | grass |

# Next Iteration?

- Your intuition will guide you, once you have gained a little bit more of the data mining experience
- Your knowledge about the methods, how they work, what they can accomplish, what kind of data they are best suited for, etc.
- Let's explore other methods (option #2)!
- Let's try a different type of a tree

# Iteration #4

- weka.classifiers.trees.J48graft -C 0.25 -M 2

  (A method for generating a grafted (pruned or unpruned) C4 tree)
- Resulting Tree:
  - Number of Leaves : 165
  - Size of the tree : 329

- Correctly Classified Instances: 96.2 %

# Iteration #5

- weka.classifiers.trees.RandomTree -K 1 -M 1.0 -S 1
  (Class for constructing a tree that considers K randomly chosen attributes at each node.)
- Resulting Tree:
  - Size of the tree : 407
- Correctly Classified Instances: 87.8667 %

# Iteration #6

- weka.classifiers.trees.BFTree -S 1 -M 2 -N 5 -C 1.0 -P POSTPRUNED

  (Method for generating best-first decision trees)
- Resulting Tree:
  - Number of Leaves : 47
  - Size of the tree : 24

- Correctly Classified Instances 95.1333 %

# Iteration #7

- weka.classifiers.trees.DecisionStump

- Correctly Classified Instances 30.4 %

# Iteration #8

- weka.classifiers.trees.LMT -I -1 -M 15 -W 0.0

  (Classifier for building 'logistic model trees', which are classification trees with logistic regression functions at the leaves)

- Number of Leaves : 9
- Size of the Tree : 17
- Correctly Classified Instances 96.8 %

# Iteration #8: Model

```
region-centroid-row <= 155
|   value-mean <= 91.4444
|   |   rawred-mean <= 24.6667
|   |   |   hue-mean <= -1.89048
|   |   |   |   hue-mean <= -2.22266: LM_1:80/480 (105)
|   |   |   |   hue-mean > -2.22266
|   |   |   |   |   rawred-mean <= 2.55556: LM_2:0/408 (139)
|   |   |   |   |   rawred-mean > 2.55556
|   |   |   |   |   |   region-centroid-row <= 121: LM_3:5/413 (87)
|   |   |   |   |   |   region-centroid-row > 121: LM_4:0/408 (36)
|   |   |   hue-mean > -1.89048: LM_5:19/339 (253)
|   |   rawred-mean > 24.6667
|   |   |   hue-mean <= -2.17742: LM_6:80/400 (22)
|   |   |   hue-mean > -2.17742: LM_7:80/400 (183)
|   value-mean > 91.4444: LM_8:0/160 (220)
region-centroid-row > 155: LM_9:80/160 (455)
```

# Iteration #8: Results

=== Confusion Matrix ===

| a | b | c | d | e | f | g | <-- classified as |
|---|---|---|---|---|---|---|---|
| 203 | 0 | 0 | 0 | 2 | 0 | 0 | a = brickface |
| 0 | 220 | 0 | 0 | 0 | 0 | 0 | b = sky |
| 0 | 0 | 193 | 3 | 11 | 0 | 1 | c = foliage |
| 2 | 0 | 1 | 208 | 8 | 1 | 0 | d = cement |
| 1 | 0 | 14 | 4 | 185 | 0 | 0 | e = window |
| 0 | 0 | 0 | 0 | 0 | 236 | 0 | f = path |
| 0 | 0 | 0 | 0 | 0 | 0 | 207 | g = grass |

# Iteration #9

- weka.classifiers.functions.MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a

  (method for the Multilayered Perceptron ANN Learning)

- Results: 96.7333 %
- Model: black box

# Iteration #10

- weka.classifiers.trees.RandomForest -I 10 -K 0 -S 1 (method for the Random Forest Learning)

- Results: 97.6 %
- Model: black box ("Random forest of 10 trees, each constructed while considering 5 random features")

# Stop?

- Yes
- Let's pick a model and fine tune the parameters
- LMT?  Really slow to build.
- Another option?
- How about wJ48graft?
  - 96.8 % vs. 96.2%
  - Faster to build
  - Easier readability

# Next step

- Fine-tuning of the parameters
- Try several options – same or lesser results

- Keep the tree

- Present it to the Research Group

# Summary

- Another start-to-end easy data mining project
- The importance of having data mining expertise and understanding of the methods and the evaluation
- Building our experience and intuition

# Conclusion

- Data Mining is a highly iterative process
- Lots of paths and lots of feedback
- Many traversing options
- Your intuition will guide you, once you have gained a little bit more of the data mining experience
- Your knowledge about the methods, how they work, what they can accomplish, what kind of data they are best suited for, etc.

# First, Let's Save Our Work

- First, save the current data file as IMAGE_AssignmentII.arff, we will use it in Lesson 4 and then Assignment II

# Next

- Next is Lesson 4: Moving on to more complex datasets and tasks