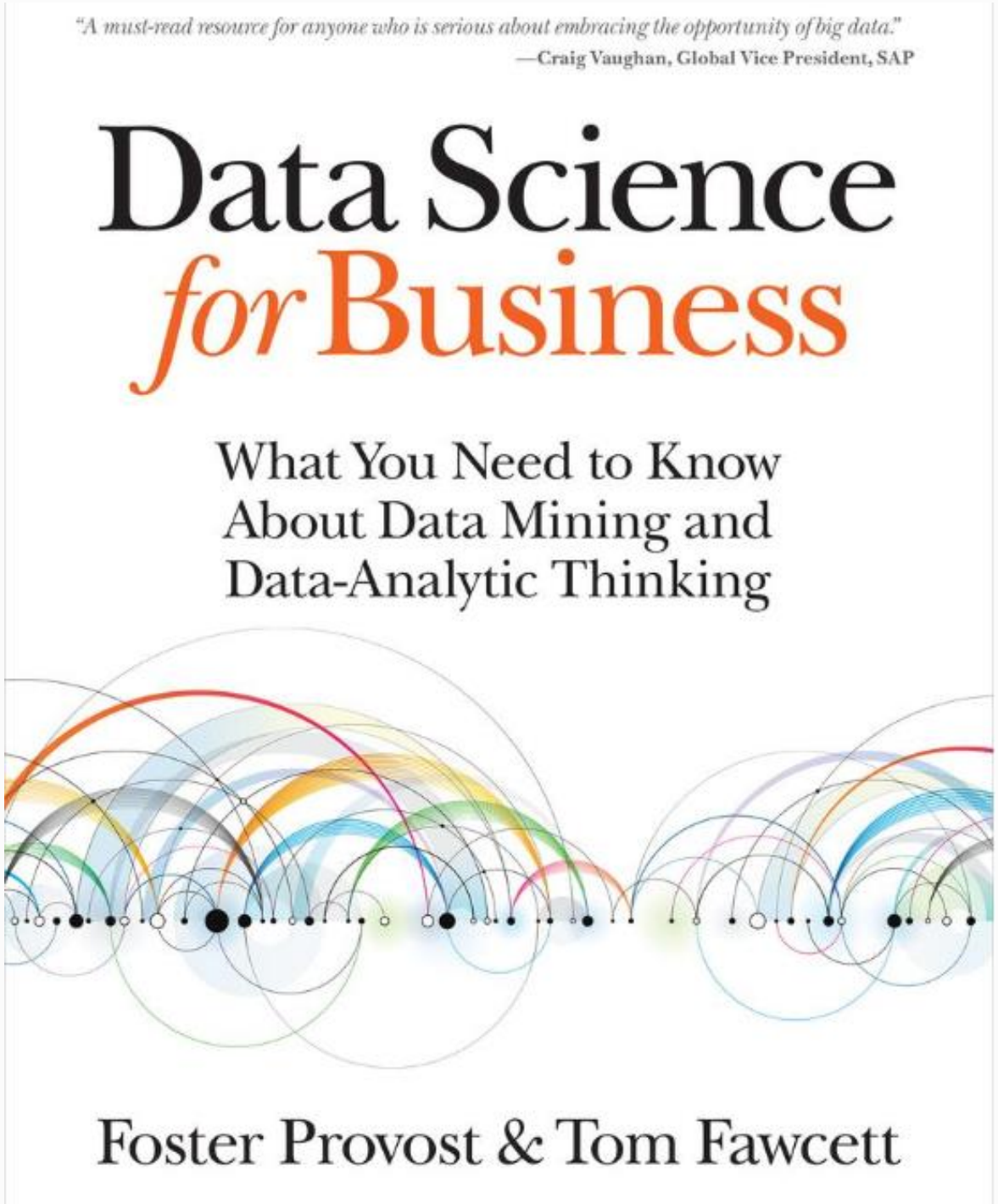
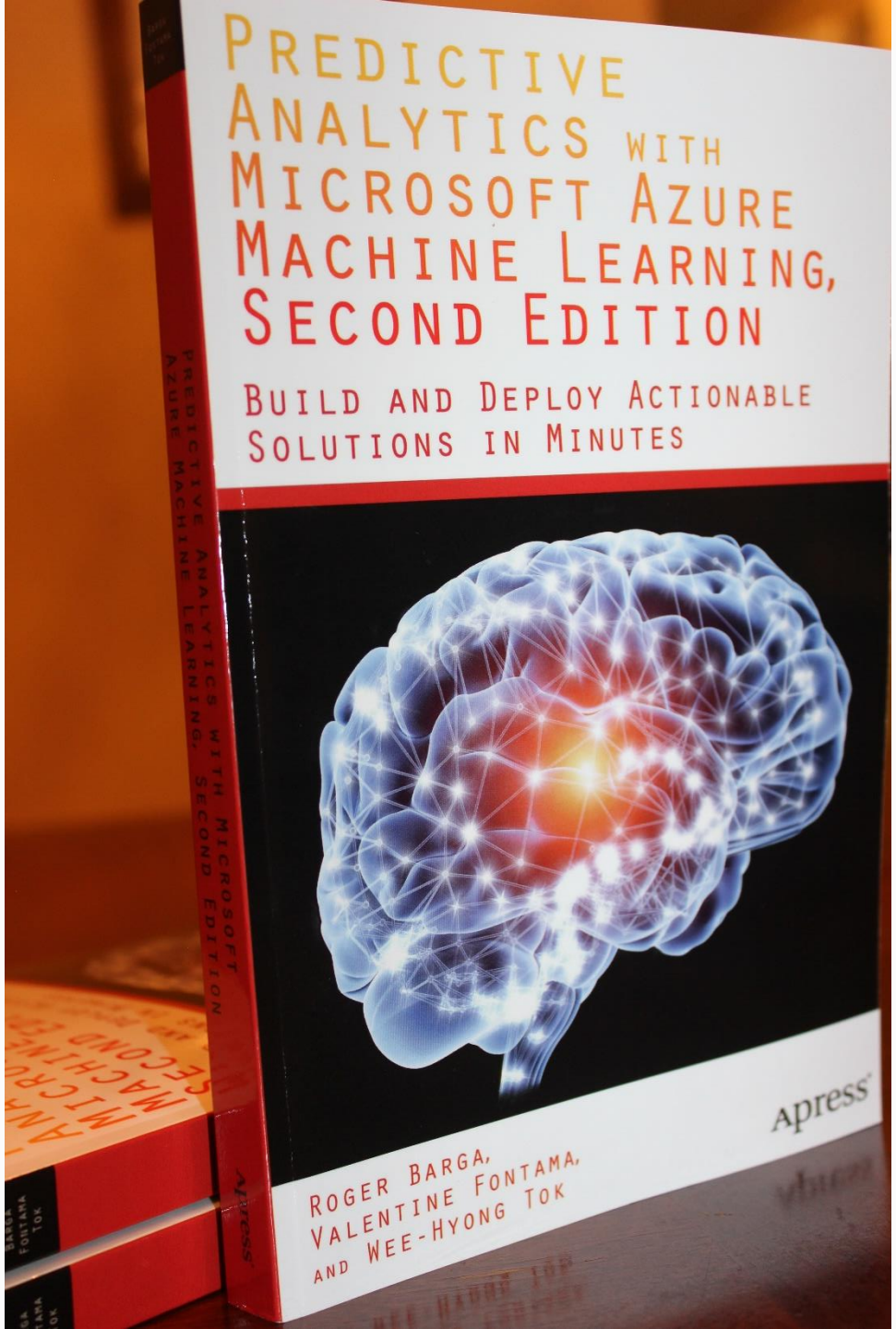
Data Science

Deriving Knowledge from Data at Scale

Nov 30th, 2015



Book Recommendation #1



Book Recommendation #2

Neural Networks

http://www.youtube.com/watch?v=GrugzF0-V3I

Machine Learning Applications for Data Center Optimization

Jim Gao, Google

Abstract

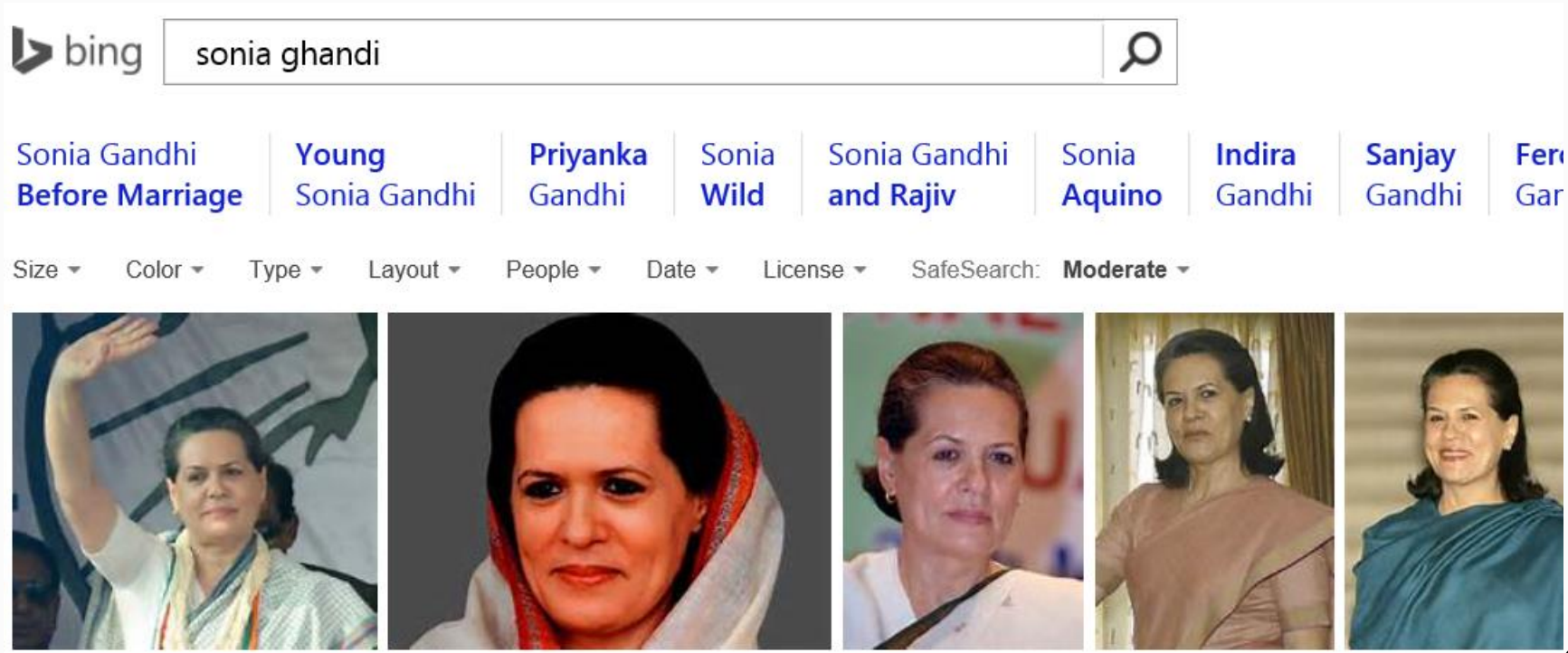
The modern data center (DC) is a complex interaction of multiple mechanical, electrical and controls systems. The sheer number of possible operating configurations and nonlinear interdependencies make it difficult to understand and optimize energy efficiency. We develop a neural network framework that learns from actual operations data to model plant performance and predict PUE within a range of 0.004 +/- 0.005 (mean absolute error +/- 1 standard deviation), or 0.4% error for a PUE of 1.1. The model has been extensively tested and validated at Google DCs. The results demonstrate that machine learning is an effective way of leveraging existing sensor data to model DC performance and improve energy efficiency.

1. Introduction

The rapid adoption of Internet-enabled devices, coupled with the shift from consumer-side computing to SaaS and cloud-based systems, is accelerating the growth of large-scale data centers (DCs). Driven by significant improvements in hardware affordability and the exponential growth of Big Data, the modern Internet company encompasses a wide range of characteristics including personalized user experiences and minimal downtime. Meanwhile, popular hosting services such as Google Cloud Platform and Amazon Web Services have dramatically reduced upfront capital and operating costs, allowing companies with smaller IT resources to scale quickly and efficiently across millions of users. These trends have resulted in the rise of large-scale DCs and their corresponding operational challenges.

One of the most complex challenges is power management. Growing energy costs and environmental responsibility have placed the DC industry under increasing pressure to improve its operational efficiency. According to Koomey, DCs comprised 1.3% of the global energy usage in 2010 [1]. At this scale, even relatively modest efficiency improvements yield significant cost savings and avert millions of tons of carbon emissions.

The first image returned is Rajiv Gandhi (her husband) in the Answer.



Query Image

Nearest Neighbor Images from the Index

Prev

DNN

Prev

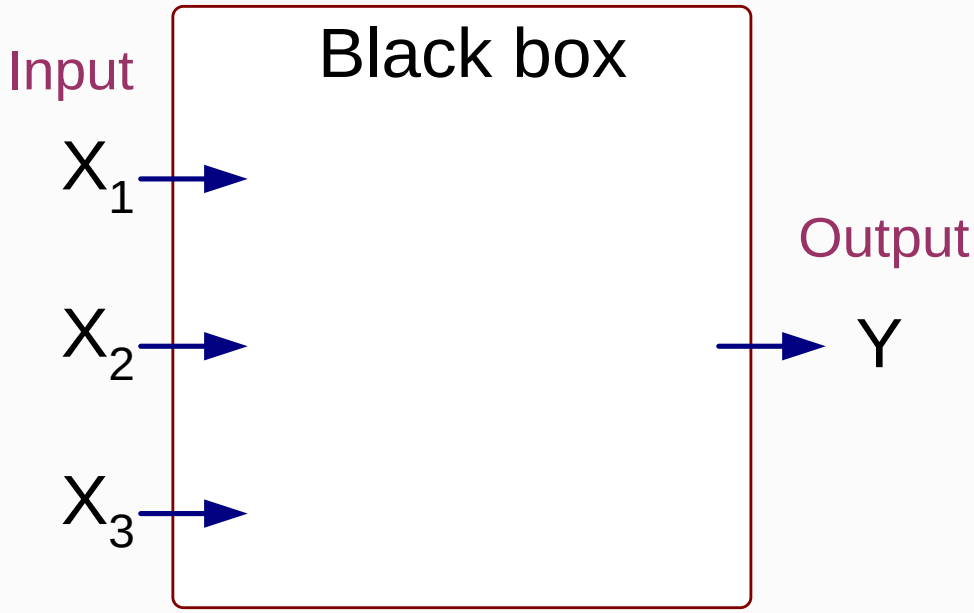
DNN

Prev

DNN

Artificial Neural Networks (ANN)

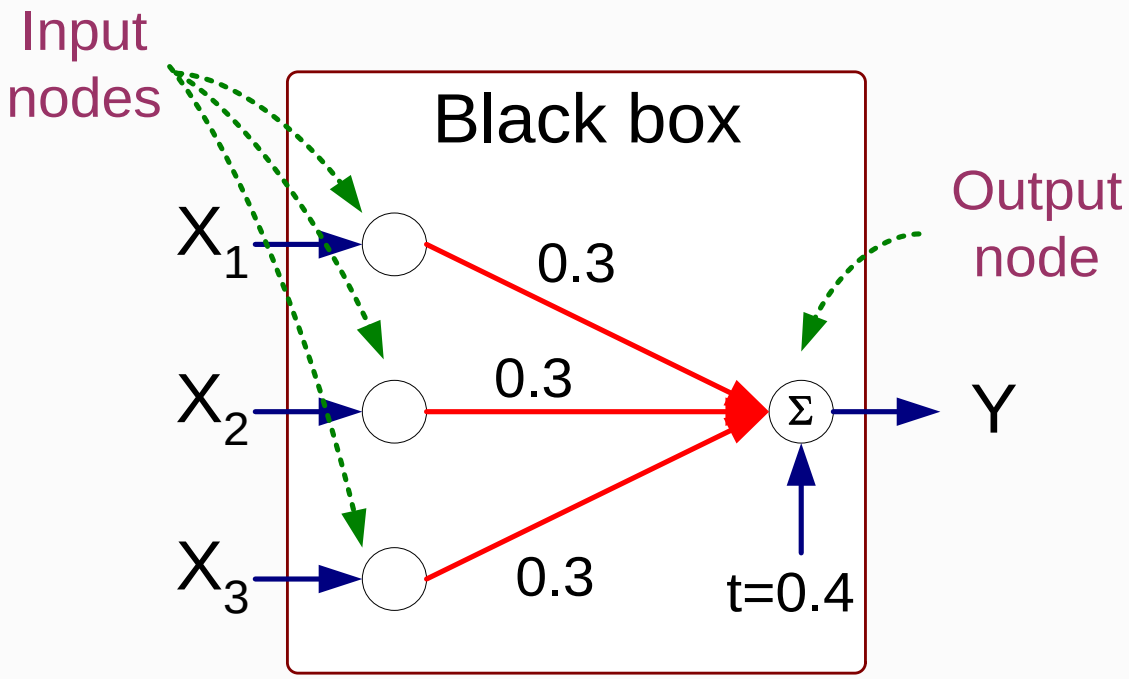
X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



Output Y is 1 if at least two of the three inputs are equal to 1.

Artificial Neural Networks (ANN)

X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0

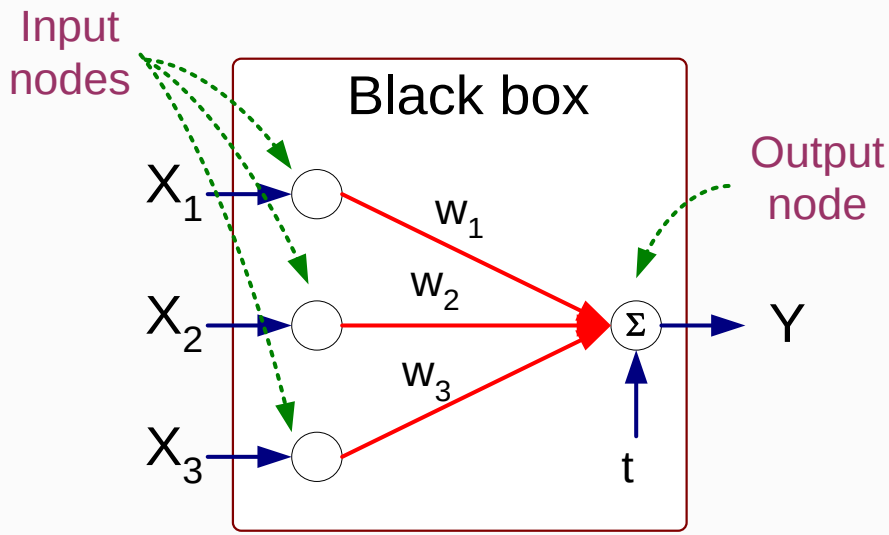


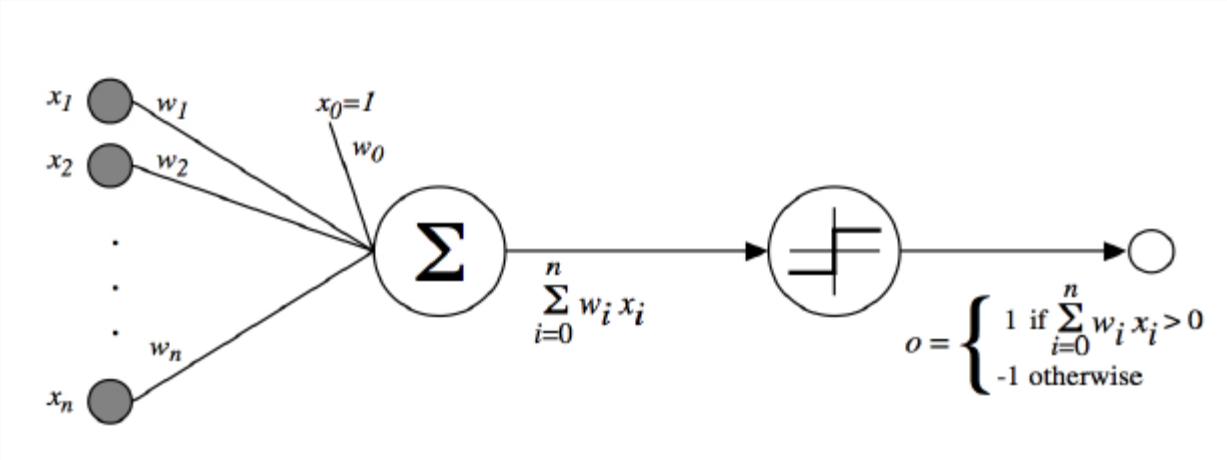
$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0)$$

$$\text{where } I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

Artificial Neural Networks (ANN)

- Model is an assembly of inter-connected nodes and weighted links;
- Output node sums up each of its input value according to the weights of its links;
- Compare output node against some threshold t





Perceptron Model

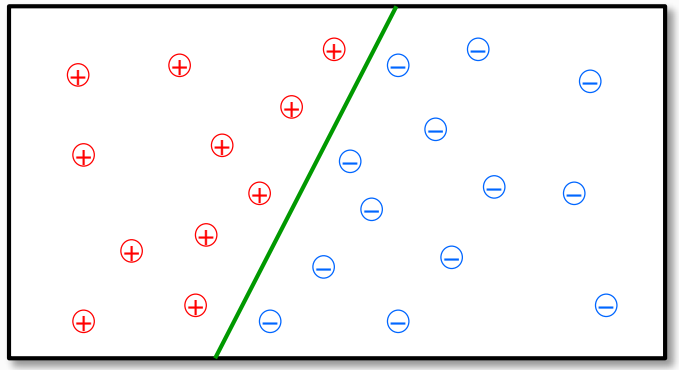
$$Y = I(\sum_i w_i X_i - t) \quad \text{or}$$

$$Y = sign(\sum_i w_i X_i - t)$$

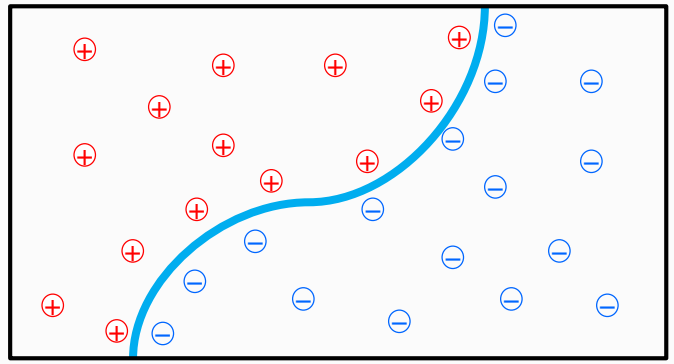
Perceptron

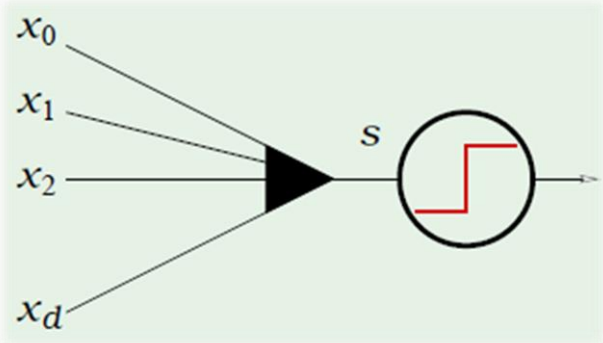
- One of the earliest ML algorithms (Rosenblatt 1958).
- On-line linear binary classification algorithm.
- Determines a hyperplane (line in $\mathbb{R}^2$, plane in $\mathbb{R}^3$,…) separating the points for the two classes.

Linearly separable data:

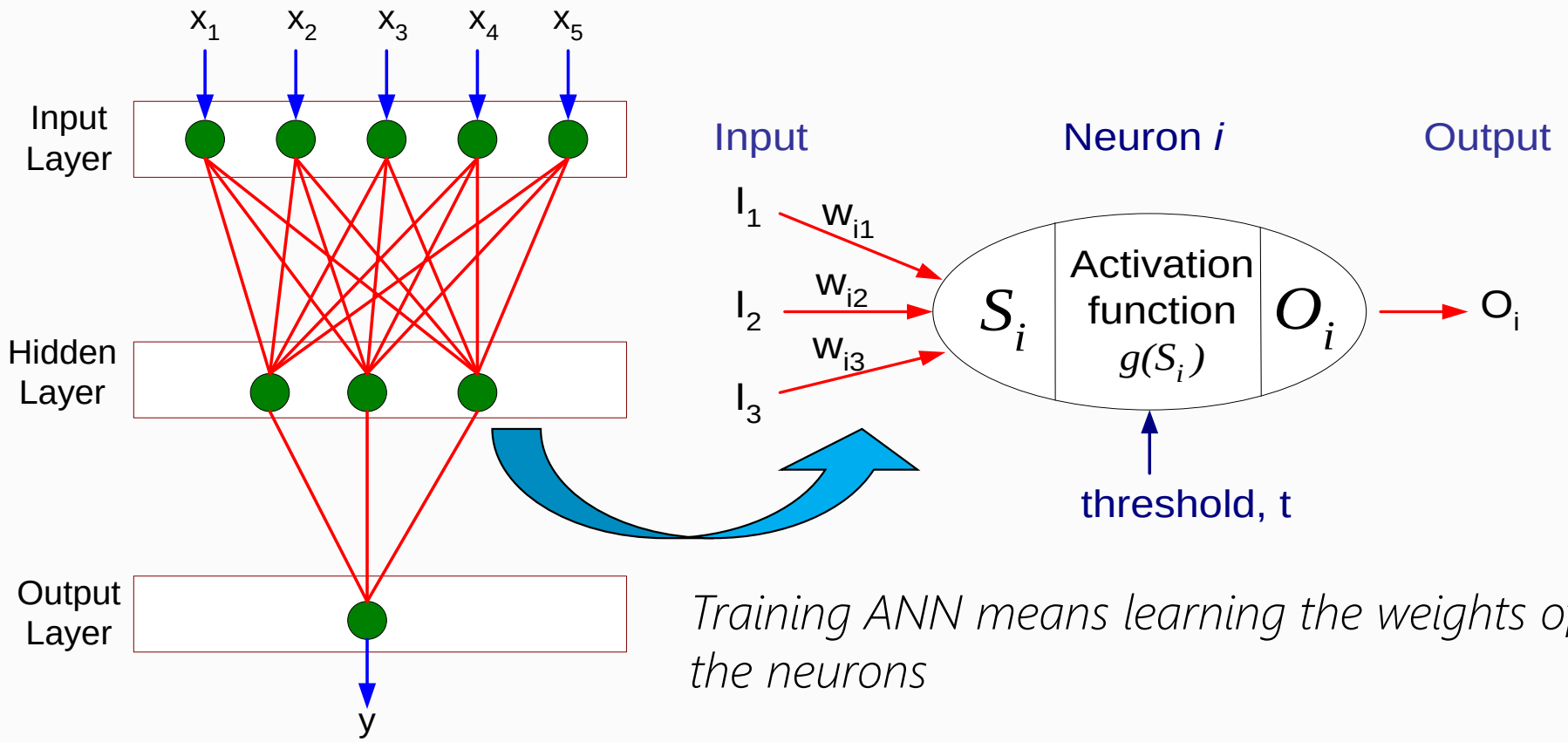


Non-linearly separable data:





General Structure of ANN

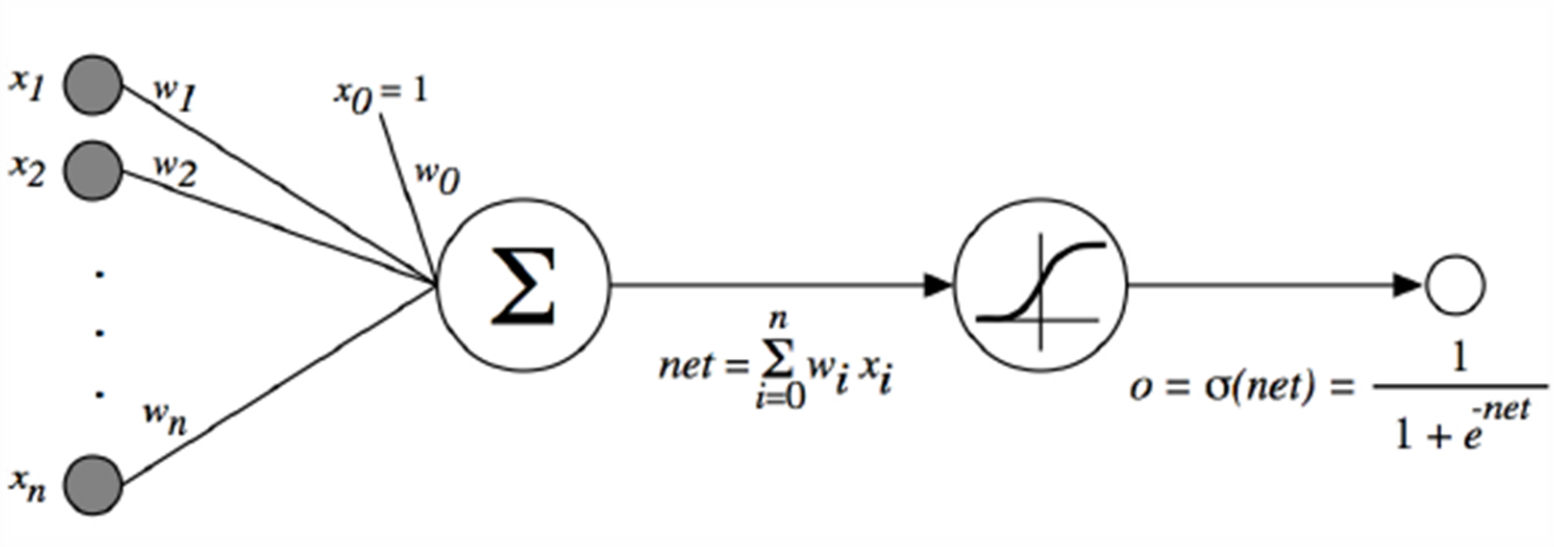


Training ANN means learning the weights of the neurons

- Perceptrons only have two layers: the input layer and the output layer
- Perceptrons only have one output unit;
- Multi-layer ANNs consist of an input layer, hidden layer, and output layer
- Multi-layer neural networks can have several output units.

Multi-Layer Artificial Neural Networks

- The units of the hidden layer function as input units to the next layer
- However, multiple layers of linear units still produce only linear functions
- The step function in perceptrons is another choice, but it is not differentiable, and therefore not suitable for gradient descent search
- Solution: the sigmoid function, a non-linear, differentiable threshold function



How A Multi-Layer Neural Network Works

- The inputs to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the input layer
- They are then weighted and fed simultaneously to a hidden layer
- The number of hidden layers is arbitrary, although usually only one
- The weighted outputs of the last hidden layer are input to units making up the output layer, which emits the network's prediction
- The network is feed-forward: None of the weights cycles back to an input unit or to an output unit of a previous layer
- From a statistical point of view, networks perform nonlinear regression: Given enough hidden units and enough training samples, they can closely approximate any function

Define the Network Topology

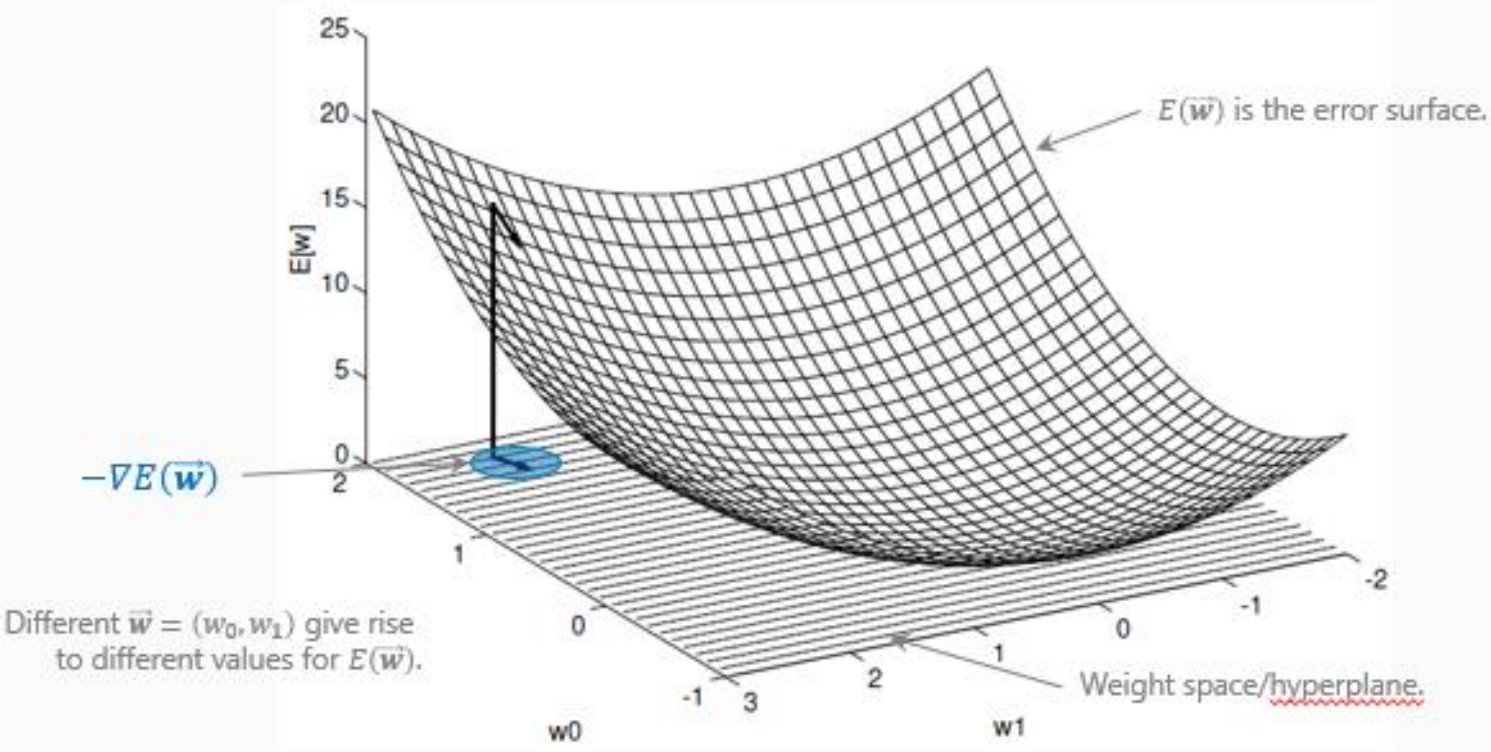
- Decide the network topology: Specify # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in *output layer*
- Normalize the input values for each attribute measured in the training tuples to [0.0 - 1.0]
- One input unit per domain value, each initialized to 0
- Output, if for classification and more than two classes, one output unit per class is used
- Once a network has been trained and its accuracy is unacceptable, repeat the training process with a *different network topology* or a *different set of initial weights*

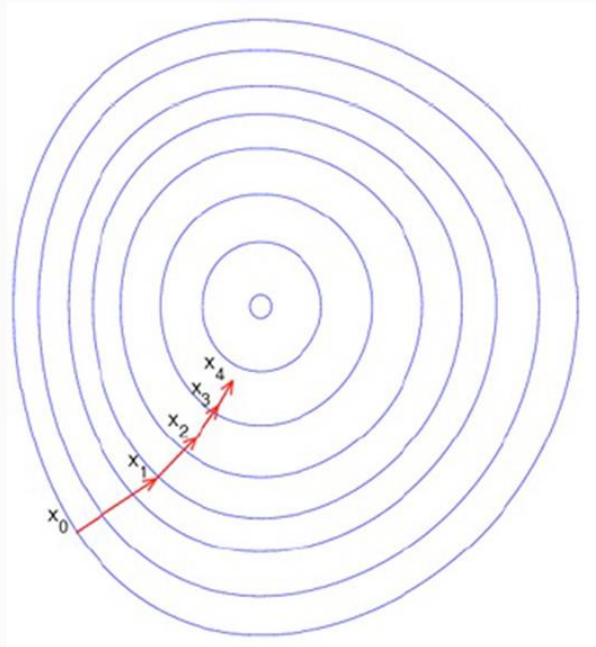
Math Fact

The gradient of the error: $\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_d}\right]$

(a vector in weight space) specifies the direction of the argument that leads to the steepest increase for the value of the error.

The negative of the gradient gives the direction of the steepest decrease.





Algorithm for Learning ANN

- Initialize the weights (w_0, w_1, …, w_k)
- Adjust the weights in such a way that the output of ANN is consistent with class labels of training examples
  - Objective function:

$$E = \sum_i \left[Y_i - f(w_i, X_i) \right]^2$$

  - Find the weights w_i's that minimize the above objective function
    - e.g., backpropagation algorithm

Backpropagation

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to minimize the mean squared error between the network's prediction and the actual target value
- Modifications are made in the "backwards" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "backpropagation"
- Steps
  - Initialize weights to small random numbers, associated with biases
  - Propagate the inputs forward (by applying activation function)
  - Backpropagate the error (by updating weights and biases)
  - Terminating condition (when error is very small, etc.)

Artificial Neural Networks

Known samples (historical data) are used to "train" the network.

Input data (x_i) are assigned weights (w_i) and combined in the "hidden" layer – like a set of linear regressions. These sets can then combined in additional layers – like regressions of regressions.

The sum of data and weights are transformed ("squashed") to the range of the training data and error is measured.

A supervised training algorithm uses output error to adjust network weights to minimize errors.

Artificial Neural Networks

Stochastic Gradient Descent

- Converging to a minimum can be quite slow (i.e. it can take thousands of steps). Increasing the learning rate can improve this but can lead to overstepping minima
- If there are multiple local minima in the error surface, gradient descent can get stuck in one of them and not find the global minimum
- Gradient descent updates weights after summing over all training examples
- Stochastic gradient descent alleviates these difficulties
  - Randomly shuffle the data set;
  - Run small batch of training samples through the algorithm;
  - Stochastic (or incremental) gradient descent updates weights incrementally after calculating error for batch;
  - Continue until end of training set is reached
  - Related to Noise, Jitter, Simulated Annealing, etc.

Practical Considerations

- A good BP net requires more than the core of the learning algorithms. Many parameters must be carefully selected to ensure a good performance.
- Although the deficiencies of BP nets cannot be completely cured, some of them can be eased by some practical means.
- Initial weights (and biases)
  - Random, [-0.05, 0.05], [-0.1, 0.1], [-1, 1]
  - Normalize weights for hidden layer (v_ij) (Nguyen-Widrow)
    - Random assign v_ij for all hidden units V_j
    - For each V_j, normalize its weight by $\boldsymbol{v_{ij}} = \boldsymbol{\beta} \cdot \boldsymbol{v_{ij}} / \left\| \boldsymbol{v_{.j}} \right\|_2$

      where $\left\| \boldsymbol{v_{.j}} \right\|_2$ is the normalization factor and $\boldsymbol{\beta} = 0.7\sqrt[n]{\boldsymbol{p}}$

      where $\boldsymbol{p} = \#\text{ of hidden nodes},\ \boldsymbol{n} = \#\text{ of input nodes}$
    - Avoid bias in weight initialization: $\left\| \boldsymbol{v_{.j}} \right\|_2 = \boldsymbol{\beta}$ after normalization

Training samples

- Quality and quantity of training samples determines the quality of learning results
- Samples must be good representatives of the problem space
    - Random sampling
    - Proportional sampling (with prior knowledge of the problem space)
- # of training patterns needed:
    - There is no theoretically idea number. Following is a rule of thumb
    - W: total # of weights to be trained (depends on net structure)

      e: desired classification error rate
    - If we have P = W/e training patterns, and we can train a net to correctly classify (1 – e/2)P of them,
    - Then this net would (in a statistical sense) be able to correctly classify a fraction of 1 – e input patterns drawn from the same sample space
    - Example: W = 80, e = 0.1, P = 800. If we can successfully train the network to correctly classify (1 – 0.1/2)*800 = 760 of the samples, we would believe that the net will work correctly 90% of time with other input.

Over-Training/Over-Fitting

- Trained net fits very well with the training samples (total error $E \approx 0$), but not with new input patterns
- Over-training may become serious if T
  - Training samples were not obtained properly
  - Training samples have noise

Control over-training for better generalization

- Cross-validation: dividing the samples into two sets
  - - 90% into training set: used to train the network
  - - 10% into test set: used to validate training results

  periodically test the trained net with test samples, stop training when test results start to deteriorating.

- Stop training early (before $E \approx 0$)
- Add noise to training samples: x:t becomes x+noise:t

Neural Network as a Classifier

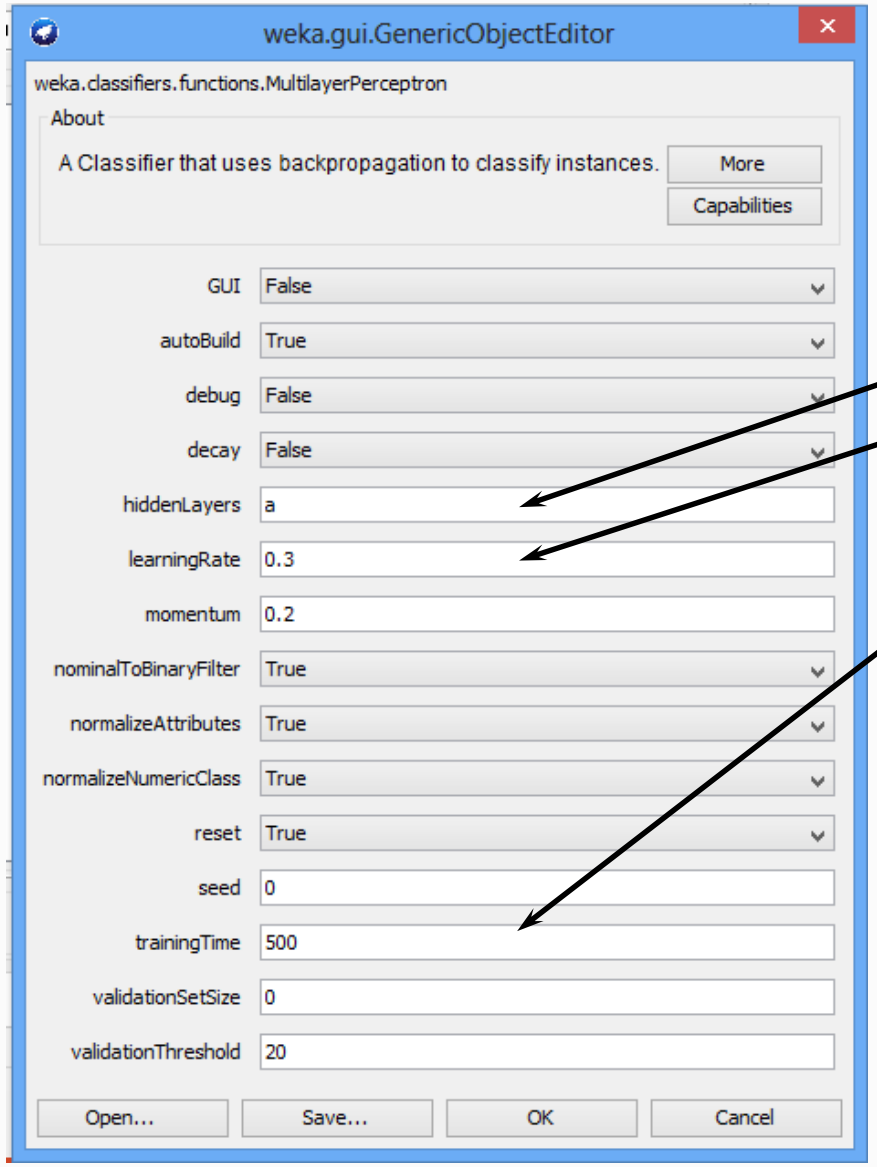
Weakness

- Long training time
- Require a number of parameters typically best determined empirically, e.g., the network topology or "structure."
- Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of "hidden units" in the network

Strength

- High tolerance to noisy data
- Ability to classify untrained patterns
- Well-suited for continuous-valued inputs *and outputs*
- Successful on an array of real-world data, e.g., hand-written letters
- Algorithms are inherently parallel
- Techniques have been developed for the extraction of rules from trained neural networks

Neural Networks in WEKA



WEKA provides user control of training parameters:

- # of hidden layers
- Learning rate
- # of iterations or epochs ("training time")
- Increment of weight adjustments in back propogation ("learning rate")
- Controls on varying changes to increments ("momentum") and weight decay

Hyperparameters

Gradient descent

- Initial weights
- Learning rate schedule
- Batch size
- Momentum
- Stopping criteria

Tuning

- Split data set into training, cross-validation (cv), and test
- Fit model on training set
- Tune hyperparameters on CV set
- Evaluate on test set

Some problems with backpropagation

The amount of information that each training case provides about the weights is at most the log of the number of possible output labels.

- So to train a big net we need lots of labeled data.

In nets with many layers of weights the backpropagated derivatives either grow or shrink multiplicatively at each layer.

- Learning is tricky either way.

Dumb gradient descent is not a good way to perform a global search for a good region of a very large, very non-linear space.

- So deep nets trained by backpropagation are rare in practice.

A solution to all of these problems

- Use greedy unsupervised learning to find a sensible set of weights one layer at a time. Then fine-tune with backpropagation.
- Greedily learning one layer at a time scales well to really deep networks.
- Most of the information in the final weights comes from modeling the distribution of input vectors.
  - The precious information in the labels is only used for the final fine-tuning.
- We do not start backpropagation until we already have sensible weights that already do well at the task.
  - So the fine-tuning is well-behaved and quite fast.

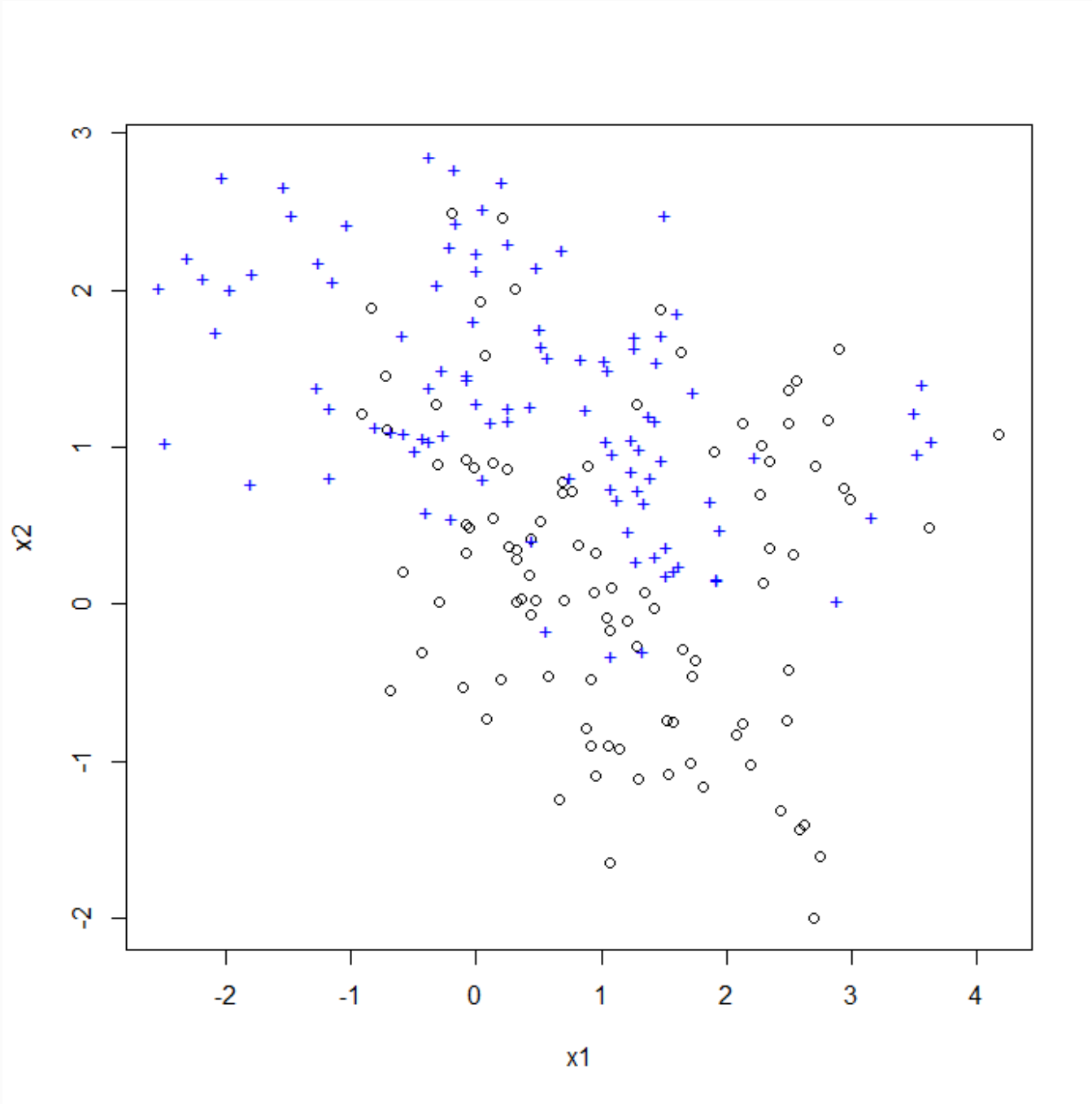
Neural Network

- Classification and Regression
- Similar to projection pursuit regression
- Very flexible, hence many “nobs to turn”
- Needs SME of the problem at hand
- Customize topography of the network
- Many ways to enhance it to avoid overfitting: bagging, boosting, Bayesian
- Powerful but requires experience to tune the model

Example:

Attributes: x1, x2
Response: y = binary (2 classes)

Apply neural network to classify these 2 classes.



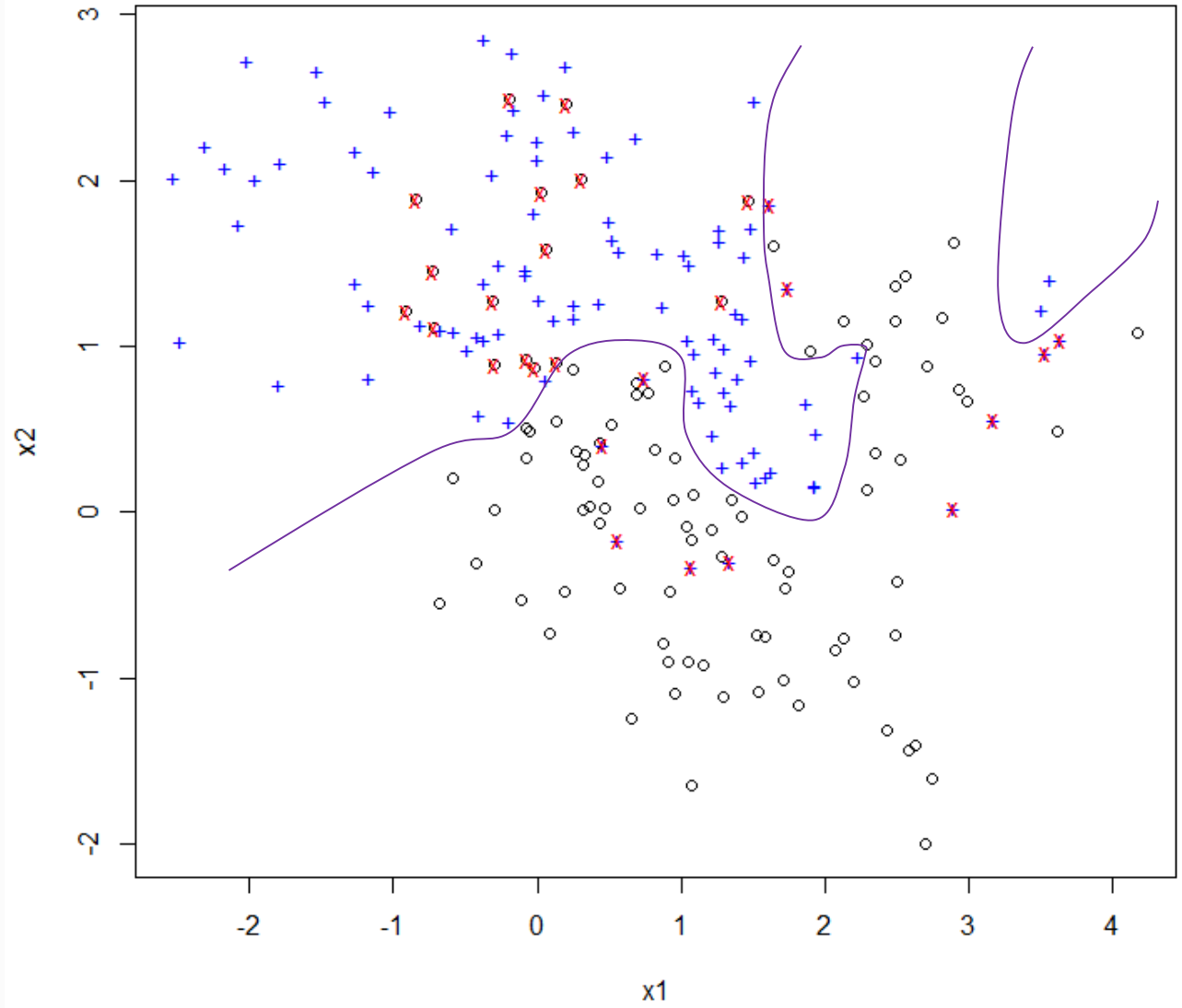
Ref: Hastie, Tibshirani and Friedman

Example

- Single hidden layer neural network with 10 nodes

- Confusion matrix:

	Predict	
	0	1
Original 0	84	16
Original 1	11	89

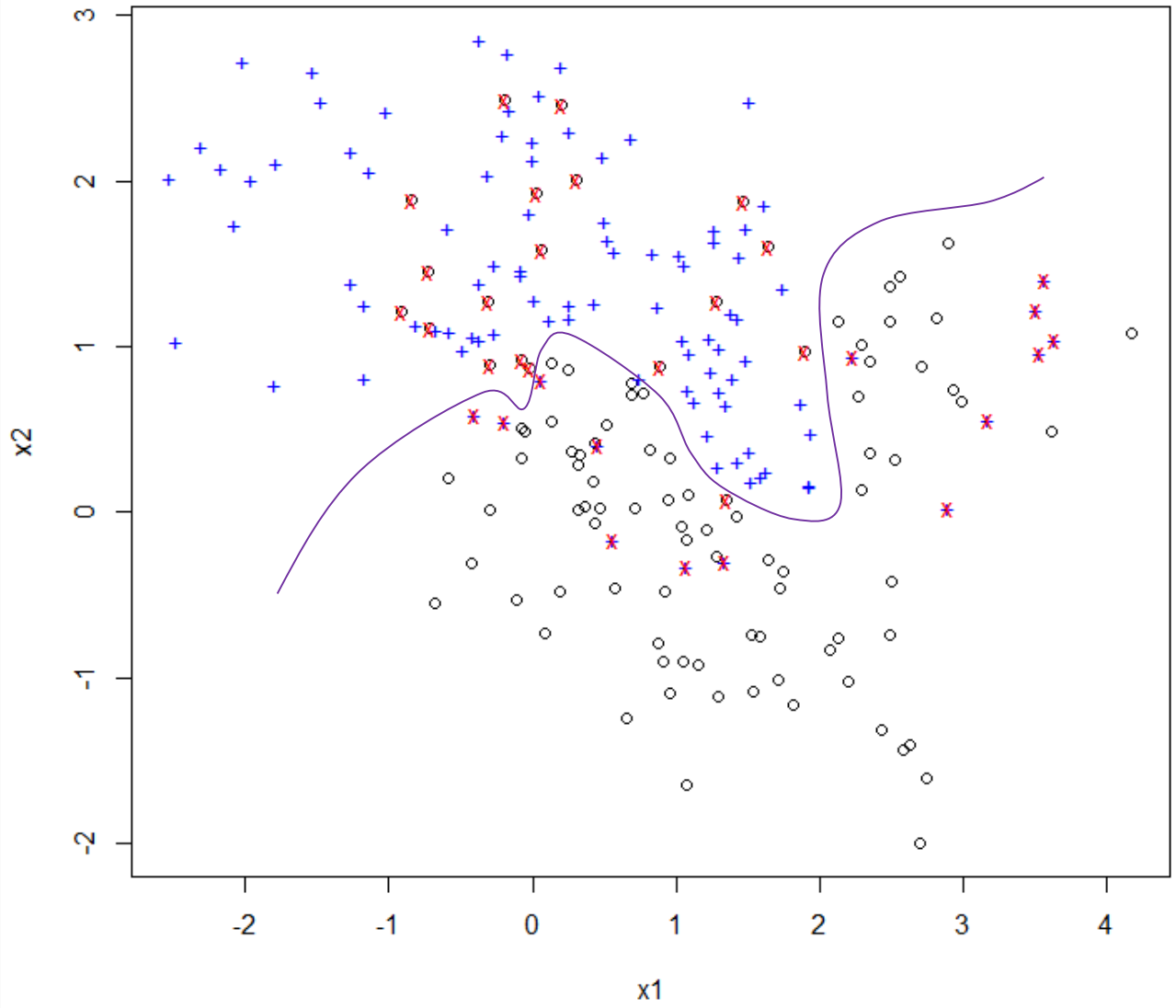


Example

- Single hidden layer neural network with 10 nodes
- Change decay parameter

- Confusion matrix:

	Predict	
	0	1
Original 0	81	19
Original 1	14	86



Break…

Association Rule

- Unsupervised technique to find relationships represented in rule
- Originated from transactional data (like retails)
- Seek patterns that lead to certain outcome
- E.g. If a purchase involves {milk, diaper} then it is likely the purchase would include {beer}.

- Use Bayes rule to define different components of the relationship

Bayes rule

- A, B are two events
- P(A) = probability of A occurs
- P(A & B) = probability of A and B occur together
- P(A | B) = probability of A occurring knowing B has occurred
- P(A | B) = P(A & B) / P(B)
- Bayes rule:
  - P(A | B) = P(A)*P(B|A)/[P(A)*P(B|A) + P(notA)*P(B|notA)]

- We have seen this in the ELISA example to illustrate the concepts of sensitivity and specificity.

- Independence:
- P(A| B) = P(A) <<< knowing B did not change P(A)

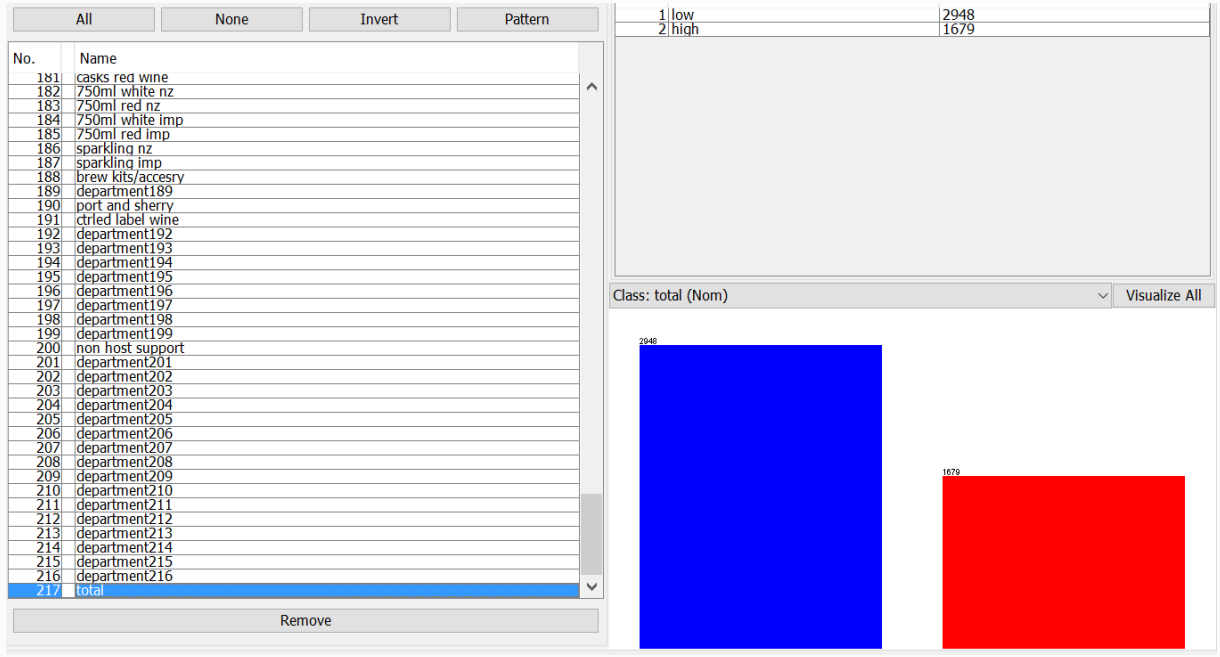
Terminology

- Support = P(event)
- Confidence = P(A | B) = prob A as a result of B
- Lift = P(A & B)/[P(A)*P(B)]

Find rules that lead to certain artist from listener's list of music

```
user,artist,sex,country
1,red hot chili peppers,f,Germany
1,the black dahlia murder,f,Germany
1,goldfrapp,f,Germany
1,dropkick murphys,f,Germany
1,le tigre,f,Germany
1,schandmaul,f,Germany
1,edguy,f,Germany
1,jack johnson,f,Germany
1,eluveitie,f,Germany
1,the killers,f,Germany
1,judas priest,f,Germany
1,rob zombie,f,Germany
1,john mayer,f,Germany
1,the who,f,Germany
1,guano apes,f,Germany
1,the rolling stones,f,Germany
3,devendra banhart,m,United States
3,boards of canada,m,United States
3,cocorosie,m,United States
3,aphex twin,m,United States
3,animal collective,m,United States
3,atmosphere,m,United States
3,joanna newsom,m,United States
3,air,m,United States
3,portishead,m,United States
3,massive attack,m,United States
3,broken social scene,m,United States
3,arcade fire,m,United States
3,plaid,m,United States
3,prefuse 73,m,United States
3,m83,m,United States
3,the flashbulb,m,United States
```

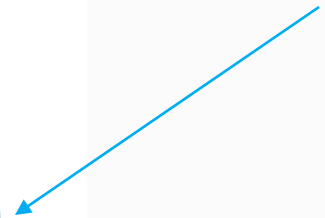
Find rules that lead to certain item purchased in a supermarket



Example: Weka>supermarket

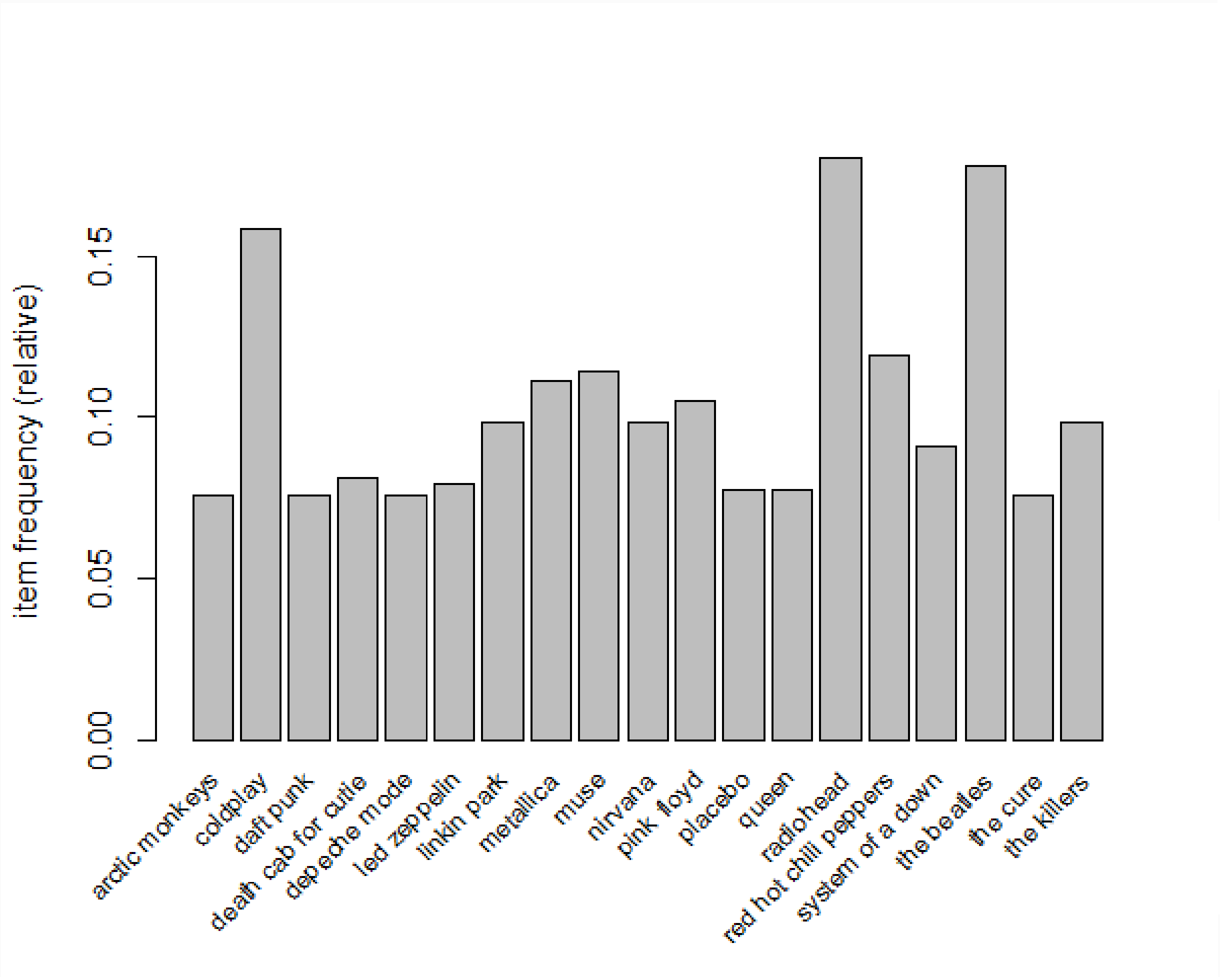
```
Scheme:       weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation:     supermarket
Instances:    4627
Attributes:   217
[list of attributes omitted]
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.15 (694 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:

Size of set of large itemsets L(1): 44

Size of set of large itemsets L(2): 380

Size of set of large itemsets L(3): 910

Size of set of large itemsets L(4): 633

Size of set of large itemsets L(5): 105

Size of set of large itemsets L(6): 1

Best rules found:

 1. biscuits=t frozen foods=t fruit=t total=high 788 ==> bread and cake=t 723    conf:(0.92)
 2. baking needs=t biscuits=t fruit=t total=high 760 ==> bread and cake=t 696    conf:(0.92)
 3. baking needs=t frozen foods=t fruit=t total=high 770 ==> bread and cake=t 705    conf:(0.92)
 4. biscuits=t fruit=t vegetables=t total=high 815 ==> bread and cake=t 746    conf:(0.92)
 5. party snack foods=t fruit=t total=high 854 ==> bread and cake=t 779    conf:(0.91)
 6. biscuits=t frozen foods=t vegetables=t total=high 797 ==> bread and cake=t 725    conf:(0.91)
 7. baking needs=t biscuits=t vegetables=t total=high 772 ==> bread and cake=t 701    conf:(0.91)
 8. biscuits=t fruit=t total=high 954 ==> bread and cake=t 866    conf:(0.91)
 9. frozen foods=t fruit=t vegetables=t total=high 834 ==> bread and cake=t 757    conf:(0.91)
10. frozen foods=t fruit=t total=high 969 ==> bread and cake=t 877    conf:(0.91)
```

{biscuit, frozen food, total=high} >>> {bread and cake}



Example: Ledolter>radio

Users= 15000
Artists= 1004




```
> inspect( sort( subset(plist.rule1, subset=lift > 4 ),by='lift' ) )
   lhs                          rhs            support    confidence lift
2  {the pussycat dolls}      => {rihanna}      0.01040000 0.5777778  13.415893
1  {t.i.}                    => {kanye west}   0.01040000 0.5672727   8.854413
5  {judas priest}            => {iron maiden}  0.01353333 0.5075000   8.562992
4  {sonata arctica}          => {nightwish}    0.01346667 0.5101010   8.236292
21 {pink floyd,the doors}    => {led zeppelin} 0.01066667 0.5387205   6.802027
20 {led zeppelin,the doors}  => {pink floyd}   0.01066667 0.5970149   5.689469
9  {megadeth}                => {metallica}    0.01626667 0.5281385   4.743759
42 {placebo,radiohead}       => {muse}         0.01366667 0.5137845   4.504247
24 {oasis,the killers}       => {coldplay}     0.01113333 0.6626984   4.180183
13 {keane}                   => {coldplay}     0.02226667 0.6374046   4.020634
17 {radiohead,snow patrol}   => {coldplay}     0.01006667 0.6344538   4.002021
```

```
> inspect( sort(subset(plist.rule1, subset=lift > 4 ), by='confidence') )
   lhs                          rhs            support    confidence lift
24 {oasis,the killers}       => {coldplay}     0.01113333 0.6626984   4.180183
13 {keane}                   => {coldplay}     0.02226667 0.6374046   4.020634
17 {radiohead,snow patrol}   => {coldplay}     0.01006667 0.6344538   4.002021
20 {led zeppelin,the doors}  => {pink floyd}   0.01066667 0.5970149   5.689469
2  {the pussycat dolls}      => {rihanna}      0.01040000 0.5777778  13.415893
1  {t.i.}                    => {kanye west}   0.01040000 0.5672727   8.854413
21 {pink floyd,the doors}    => {led zeppelin} 0.01066667 0.5387205   6.802027
9  {megadeth}                => {metallica}    0.01626667 0.5281385   4.743759
42 {placebo,radiohead}       => {muse}         0.01366667 0.5137845   4.504247
4  {sonata arctica}          => {nightwish}    0.01346667 0.5101010   8.236292
5  {judas priest}            => {iron maiden}  0.01353333 0.5075000   8.562992
```

Show artist with support > 0.075

Example: Titanic

- 2201 instances with 4 attributes
- Passenger class, gender, age group, survival outcome

```
> summary( titanic.raw )
Class       Sex         Age       Survived
1st :325   Female: 470   Adult:2092   No :1490
2nd :285   Male  :1731   Child: 109   Yes: 711
3rd :706
Crew:885

> table( titanic.raw$Class, titanic.raw$Survived )
     No Yes
1st  122 203
2nd  167 118
3rd  528 178
Crew 673 212
>
> table( titanic.raw$Sex, titanic.raw$Survived )
        No  Yes
Female  126  344
Male   1364  367
>
> table( titanic.raw$Age, titanic.raw$Survived )
      No  Yes
Adult 1438  654
Child   52   57
```

Vanilla "arules"

```
> inspect( titanic.rules )
   lhs                                rhs             support    confidence lift
1  {}                              => {Age=Adult}     0.9504771  0.9504771  1.0000000
2  {Class=2nd}                     => {Age=Adult}     0.1185825  0.9157895  0.9635051
3  {Class=1st}                     => {Age=Adult}     0.1449341  0.9815385  1.0326798
4  {Sex=Female}                    => {Age=Adult}     0.1930940  0.9042553  0.9513700
5  {Class=3rd}                     => {Age=Adult}     0.2848705  0.8881020  0.9343750
6  {Survived=Yes}                  => {Age=Adult}     0.2971377  0.9198312  0.9677574
7  {Class=Crew}                    => {Sex=Male}      0.3916402  0.9740113  1.2384742
8  {Class=Crew}                    => {Age=Adult}     0.4020900  1.0000000  1.0521033
9  {Survived=No}                   => {Sex=Male}      0.6197183  0.9154362  1.1639949
10 {Survived=No}                   => {Age=Adult}     0.6533394  0.9651007  1.0153856
11 {Sex=Male}                      => {Age=Adult}     0.7573830  0.9630272  1.0132040
12 {Sex=Female,Survived=Yes}       => {Age=Adult}     0.1435711  0.9186047  0.9664669
13 {Class=3rd,Sex=Male}            => {Survived=No}   0.1917310  0.8274510  1.2222950
14 {Class=3rd,Survived=No}         => {Age=Adult}     0.2162653  0.9015152  0.9484870
15 {Class=3rd,Sex=Male}            => {Age=Adult}     0.2099046  0.9058824  0.9530818
16 {Sex=Male,Survived=Yes}         => {Age=Adult}     0.1535666  0.9209809  0.9689670
17 {Class=Crew,Survived=No}        => {Sex=Male}      0.3044071  0.9955423  1.2658514
```

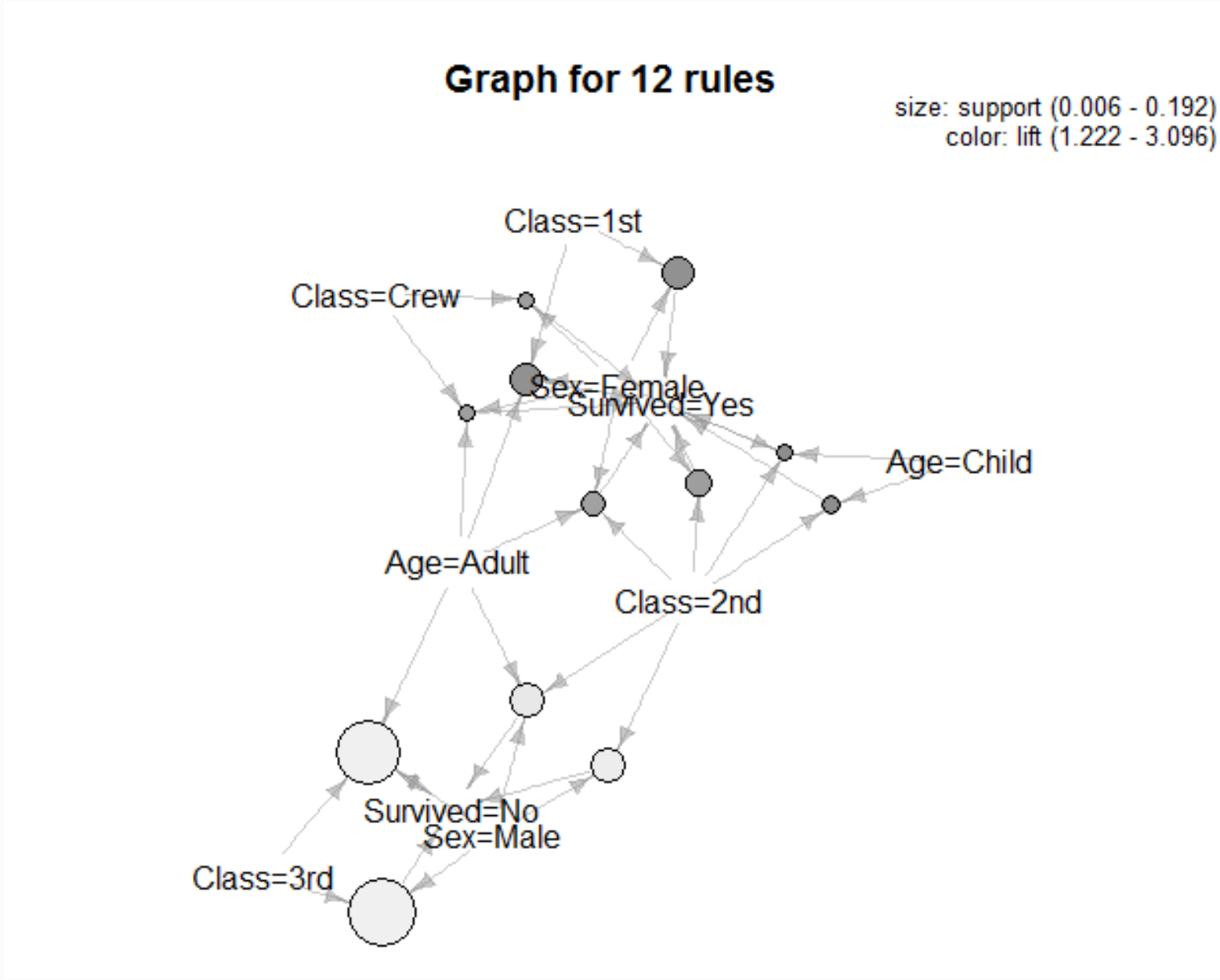

Example: titanic

More focus "arules"

```
titanic.rules2 <- apriori(titanic.raw,
                          parameter = list(minlen=2, supp=0.005, conf=0.8),
                          appearance = list(rhs=c("Survived=No", "Survived=Yes"),
                          default="lhs"),
                          control = list(verbose=F))

titanic.rules2.sorted <- sort(titanic.rules2, by="lift")

inspect(titanic.rules2.sorted )
```

```
> inspect(titanic.rules2.sorted )
lhs                                    rhs              support     confidence lift
1  {Class=2nd,Age=Child}               => {Survived=Yes} 0.010904134 1.0000000  3.095640
7  {Class=2nd,Sex=Female,Age=Child}    => {Survived=Yes} 0.005906406 1.0000000  3.095640
4  {Class=1st,Sex=Female}              => {Survived=Yes} 0.064061790 0.9724138  3.010243
10 {Class=1st,Sex=Female,Age=Adult}    => {Survived=Yes} 0.063607451 0.9722222  3.009650
2  {Class=2nd,Sex=Female}              => {Survived=Yes} 0.042253521 0.8773585  2.715986
5  {Class=Crew,Sex=Female}             => {Survived=Yes} 0.009086779 0.8695652  2.691861
11 {Class=Crew,Sex=Female,Age=Adult}   => {Survived=Yes} 0.009086779 0.8695652  2.691861
8  {Class=2nd,Sex=Female,Age=Adult}    => {Survived=Yes} 0.036347115 0.8602151  2.662916
9  {Class=2nd,Sex=Male,Age=Adult}      => {Survived=No}  0.069968196 0.9166667  1.354083
3  {Class=2nd,Sex=Male}                => {Survived=No}  0.069968196 0.8603352  1.270871
12 {Class=3rd,Sex=Male,Age=Adult}      => {Survived=No}  0.175829169 0.8376623  1.237379
6  {Class=3rd,Sex=Male}                => {Survived=No}  0.191731031 0.8274510  1.222295
```

Example: titanic



Ensemble Learning…

Combining Multiple Learners,

Resources to continue your learning, best I can find…

Ensemble-Based Systems in Decision Making
- Robi Polikar, IEEE CSM 2006

Popular Ensemble Methods: An Empirical Study
- Opitz and Maclin, Journal of AI Research 1999 (html version)

Ensemble Methods in Machine Learning
- Tom Dietterich, 2000

Ensemble Diversity Creation Methods: A Survey and Categorization
- Brown, Wyatt, Harris and Yao, Information Fusion 2005

The Boosting Approach to Machine Learning: An Overview
- Rob Schapire, 2002

Various Boosting tutorials
- Maintained by Rob Schapire

Rationale

- No Free Lunch theorem: There is no algorithm that is always the most accurate
- Generate a group of base-learners which when combined has higher accuracy
- Different learners can use different
  - Algorithms
  - Same algorithm, different hyperparameters
  - Representations (Modalities)
  - Training sets
  - Subproblems

Mathematical Intuition

Majority vote

Suppose we have 5 completely independent classifiers, then 3, 4 or 5 classifiers would have to vote for the answer.

- If accuracy is 70% for each
  - 10 $(.7^3)(.3^2)+5(.7^4)(.3)+(.7^5)$
  - **83.7% majority vote accuracy**
- 101 such classifiers
  - **99.9% majority vote accuracy**

Two Common Strategies

Bagging

- Use different samples or attributes of the examples to generate diverse classifiers
- Use voting (Average or median with regression), unstable algorithms profit from bagging
- Best Example: Random Forest

Boosting

- Make examples currently misclassified more important (or less, in some cases)
- Best Example: AdaBoost

Build Base Classifier: Bagging

JRip is the Weka implementation of the algorithm Ripper*k*. This algorithm uses incremental reduced-error pruning to build a set of classification rules.

The dataset contains 27 ligands of Acetylcholinesterase (AchE) and 1000 decoy compounds chosen from the BioInfo database.

- Molecular files for training: train-ache-t3ABl2u3.arff
- Molecular files for testing: test-ache-t3ABl2u3.arff

1. Weka, click on the button Explorer.
2. Open File, select train-ache-t3ABl2u3.arff.
3. Click on the tab Classify.
4. Test options, select Supplied test set and click
5. Select the test-ache-t3ABl2u3.arff file.
6. Close.

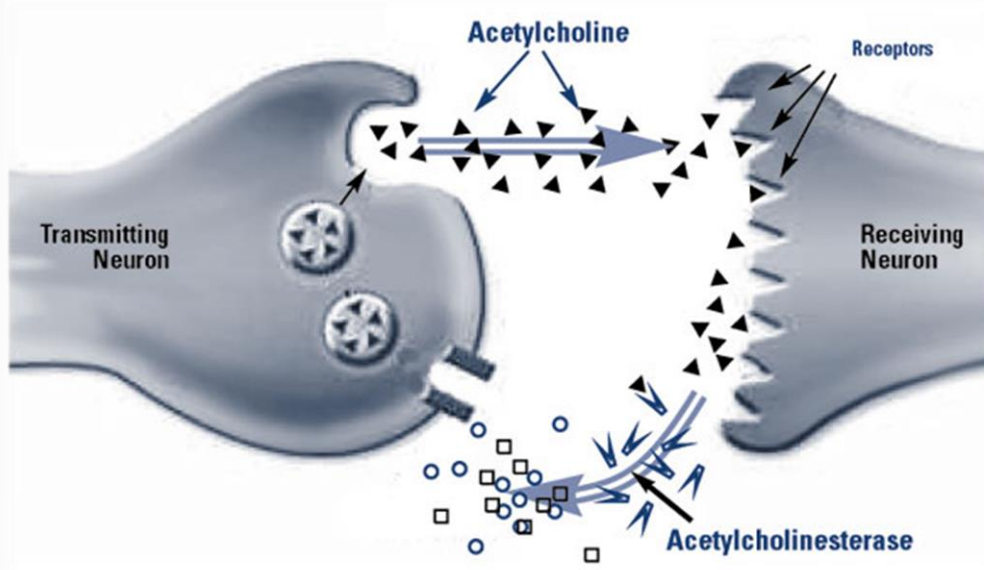
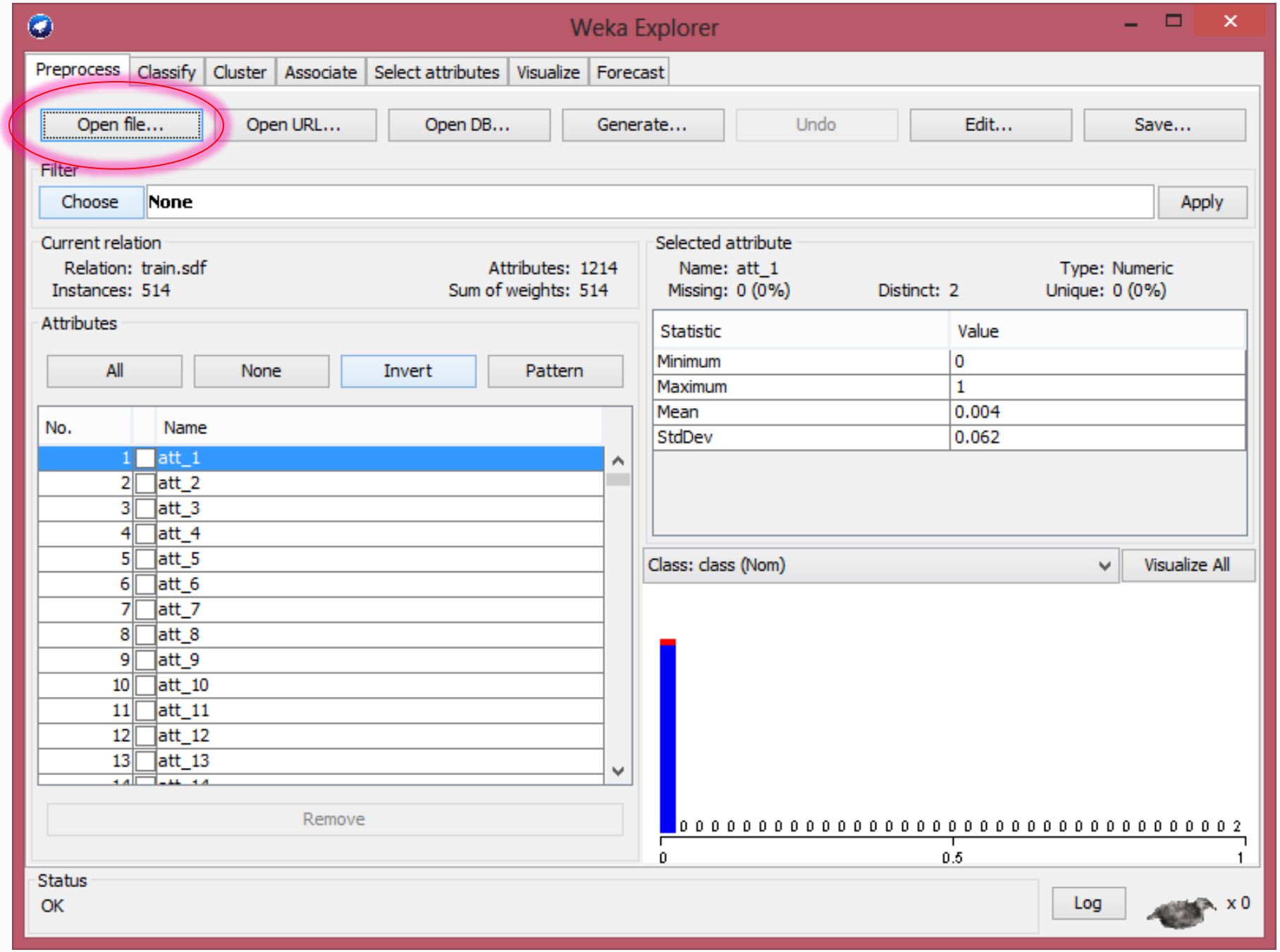
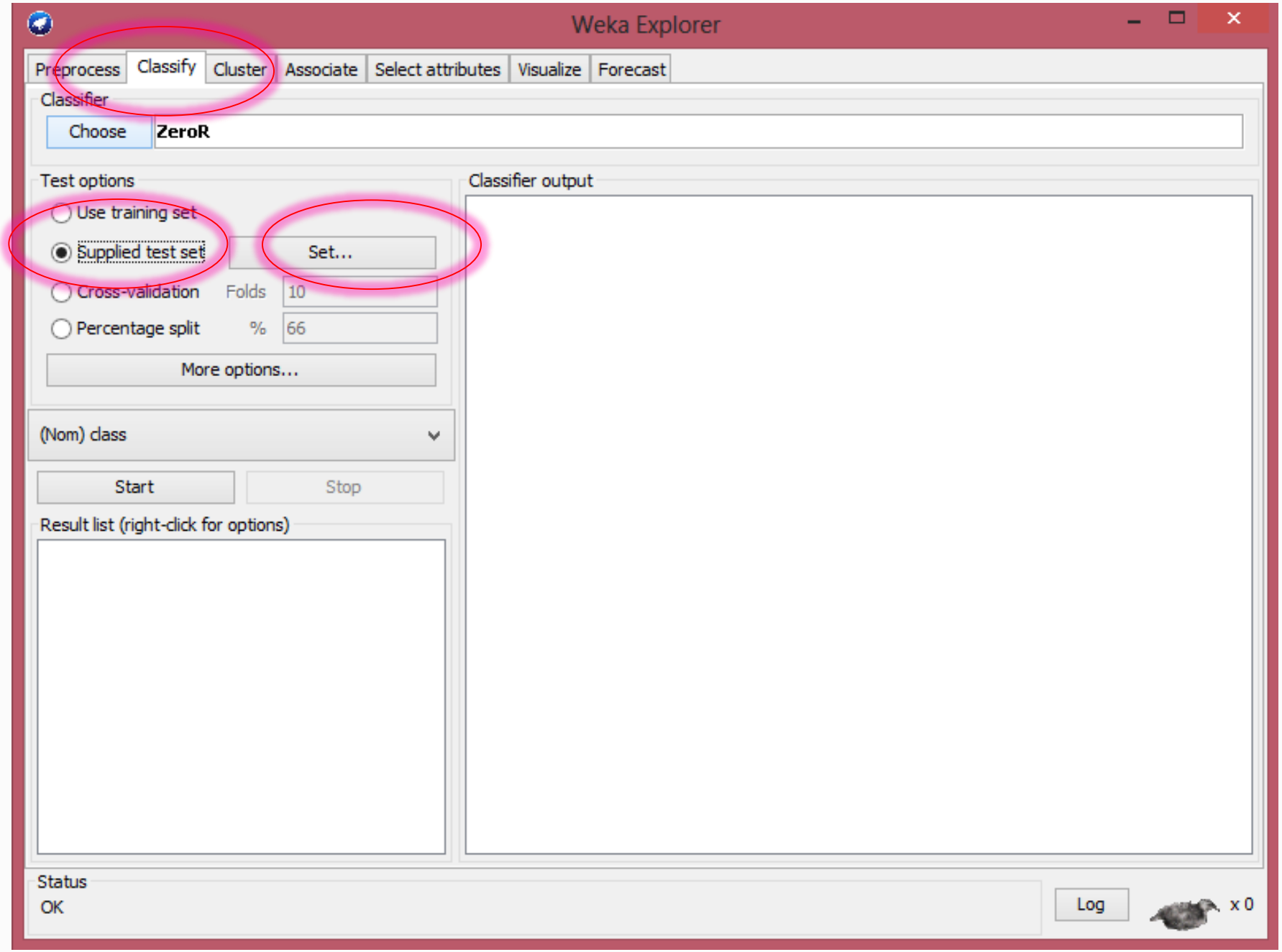


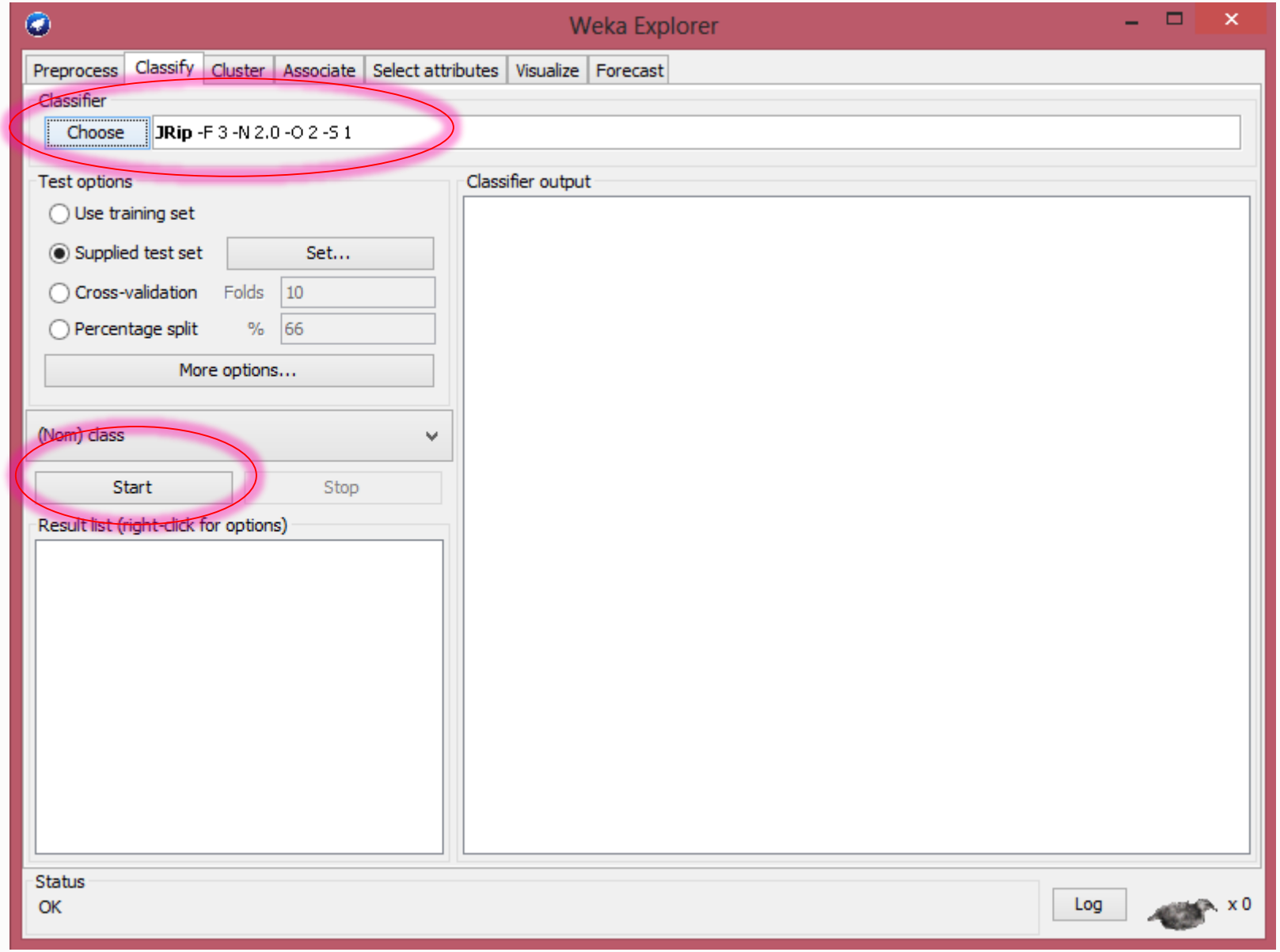
Fig. 1. After signalling, acetylcholine is released from receptors and broken down by acetylcholinesterase to be recycled in a continuous process.



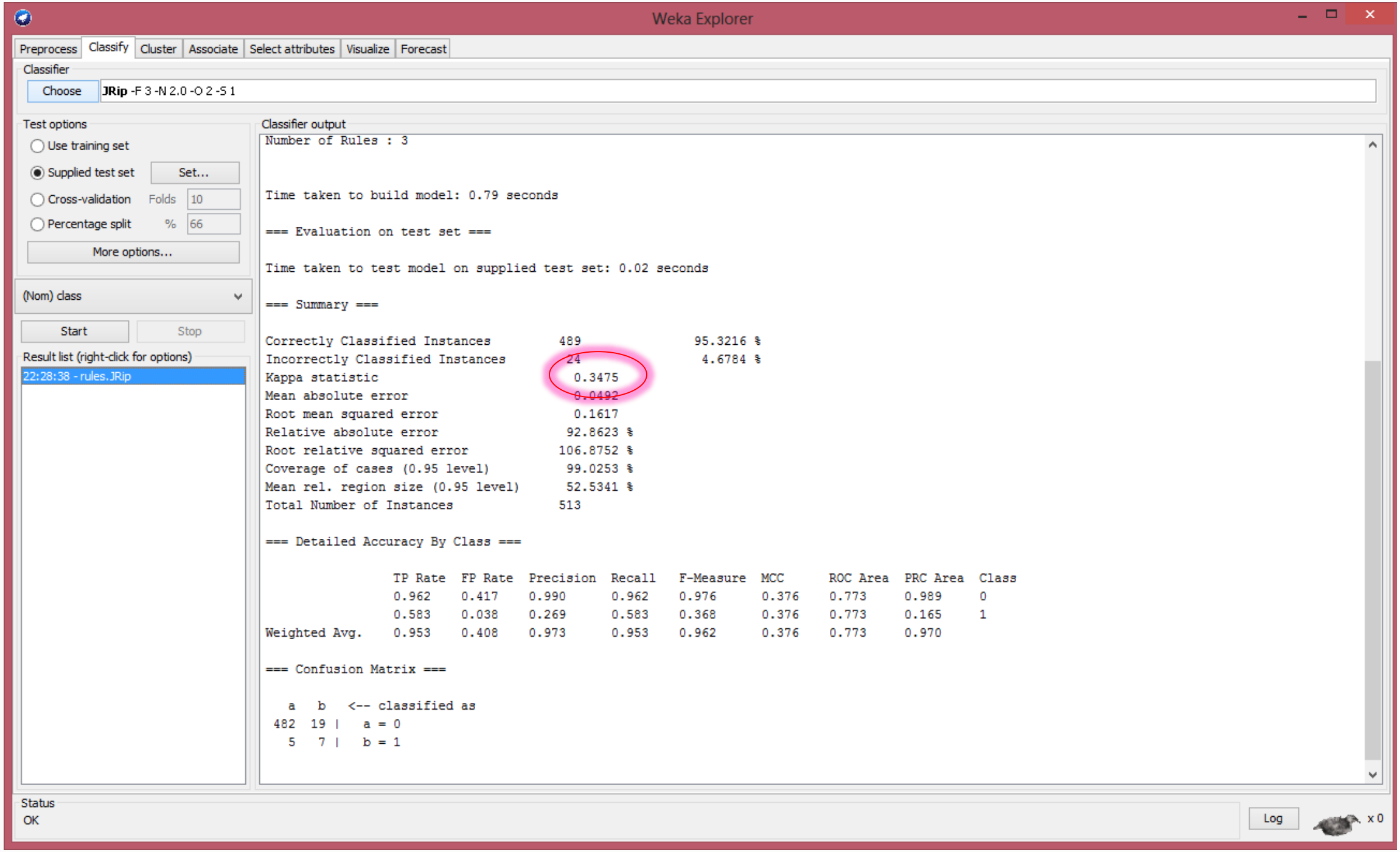
1. Preprocess
2. Open file...
3. Select training file.



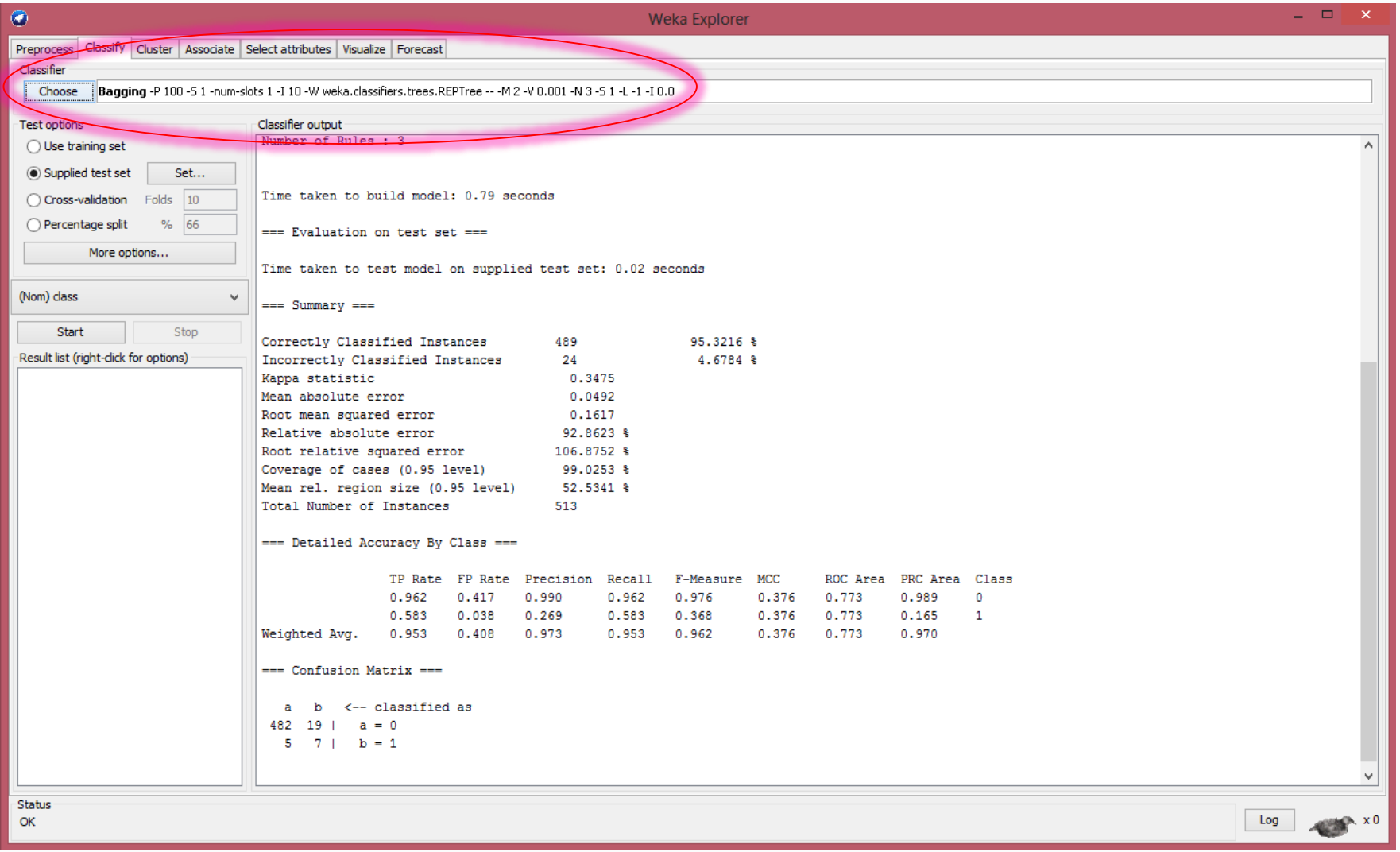
1. Classify
2. Supplied test set
3. Set
4. Select testset.arff
5. close...



1. Classify
2. Select JRip
3. Keep default parameters
4. Start


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Forecast
Classifier
Choose
JRip -F 3 -N 2.0 -O 2 -S 1
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
22:28:38 - rules.JRip
Classifier output
Number of Rules : 3
Time taken to build model: 0.79 seconds
=== Evaluation on test set ===
Time taken to test model on supplied test set: 0.02 seconds
=== Summary ===
Correctly Classified Instances 489 95.3216 %
Incorrectly Classified Instances 24 4.6784 %
Kappa statistic 0.3475
Mean absolute error 0.0492
Root mean squared error 0.1617
Relative absolute error 92.8623 %
Root relative squared error 106.8752 %
Coverage of cases (0.95 level) 99.0253 %
Mean rel. region size (0.95 level) 52.5341 %
Total Number of Instances 513
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class
0.962 0.417 0.990 0.962 0.976 0.376 0.773 0.989 0
0.583 0.038 0.269 0.583 0.368 0.376 0.773 0.165 1
Weighted Avg. 0.953 0.408 0.973 0.953 0.962 0.376 0.773 0.970
=== Confusion Matrix ===
a b <-- classified as
482 19 | a = 0
5 7 | b = 1
Status
OK
Log
x 0

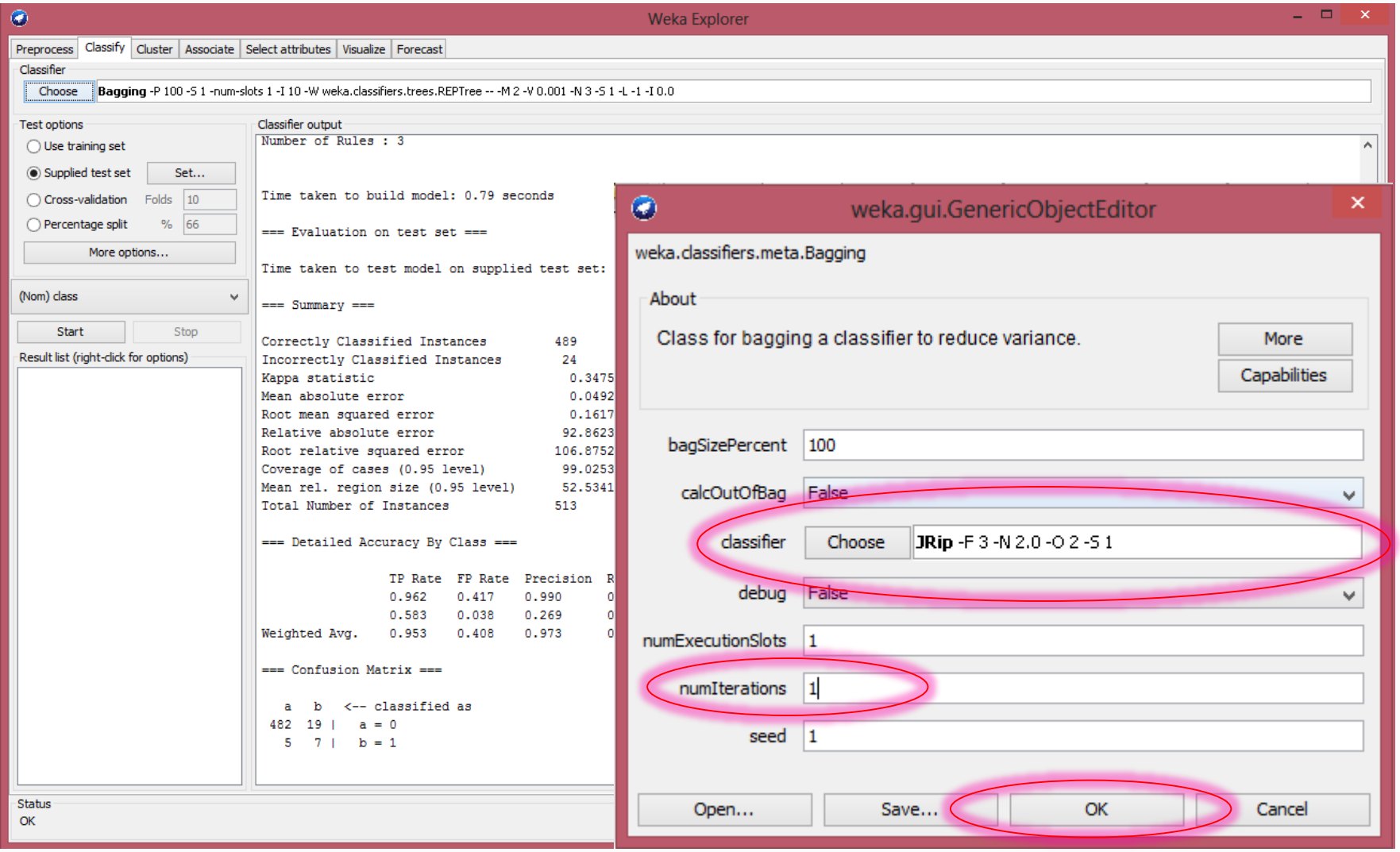
Bagging



Prepare one model

1. Click Classify, then Choose.
2. Select classifiers->meta->Bagging.
3. Click on the name of the method to the right of the Choose button.

Bagging

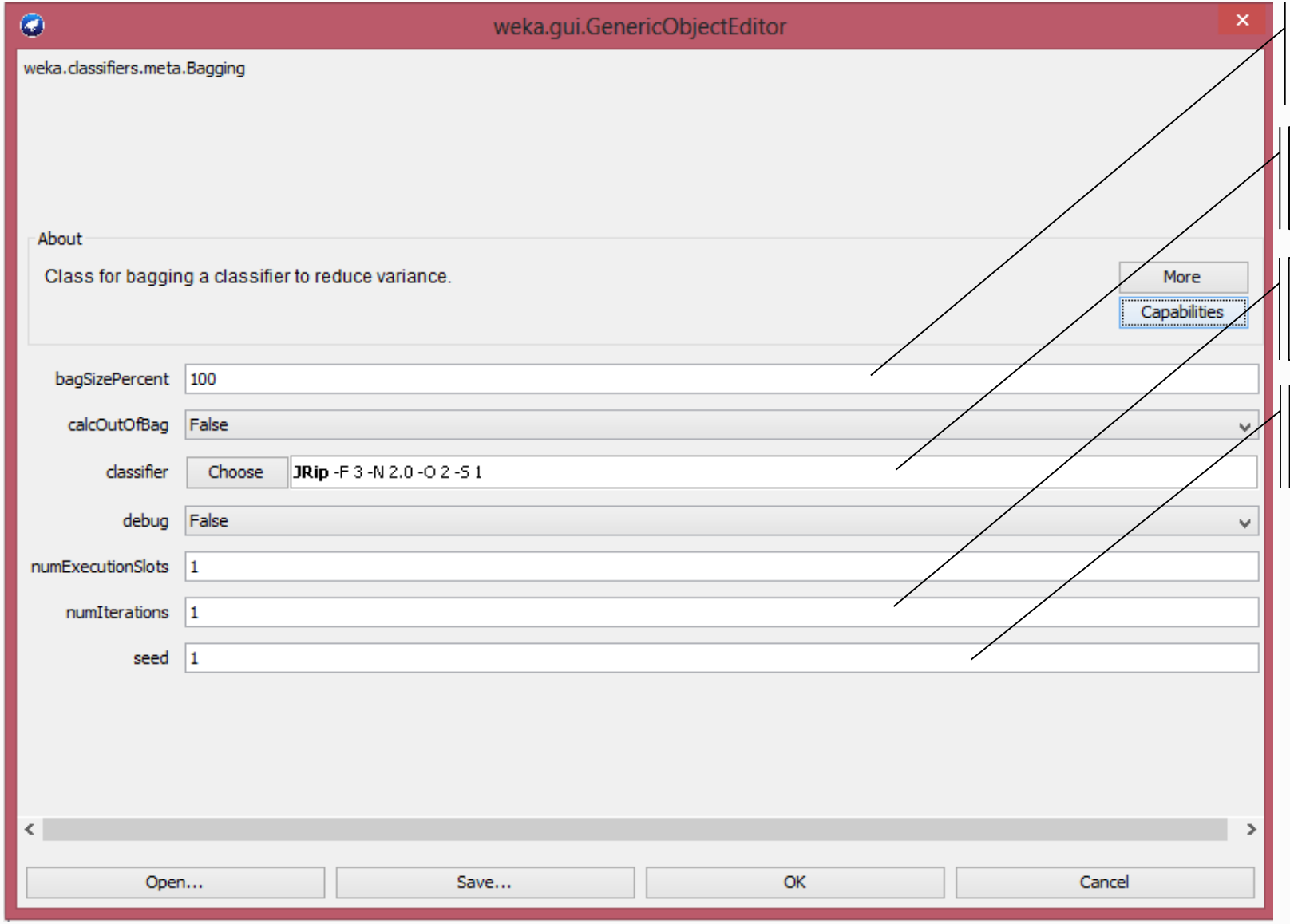


Prepare one model

1. Click **Classify**, then **Choose**.
2. Select **classifiers->meta->Bagging**.
3. Click on the name of the method to the right of the Choose button.

4. click **Choose** then select **classifiers->rules->JRip**. Set the numIterations to 1 and click **OK**

Quick double-click on bagging…



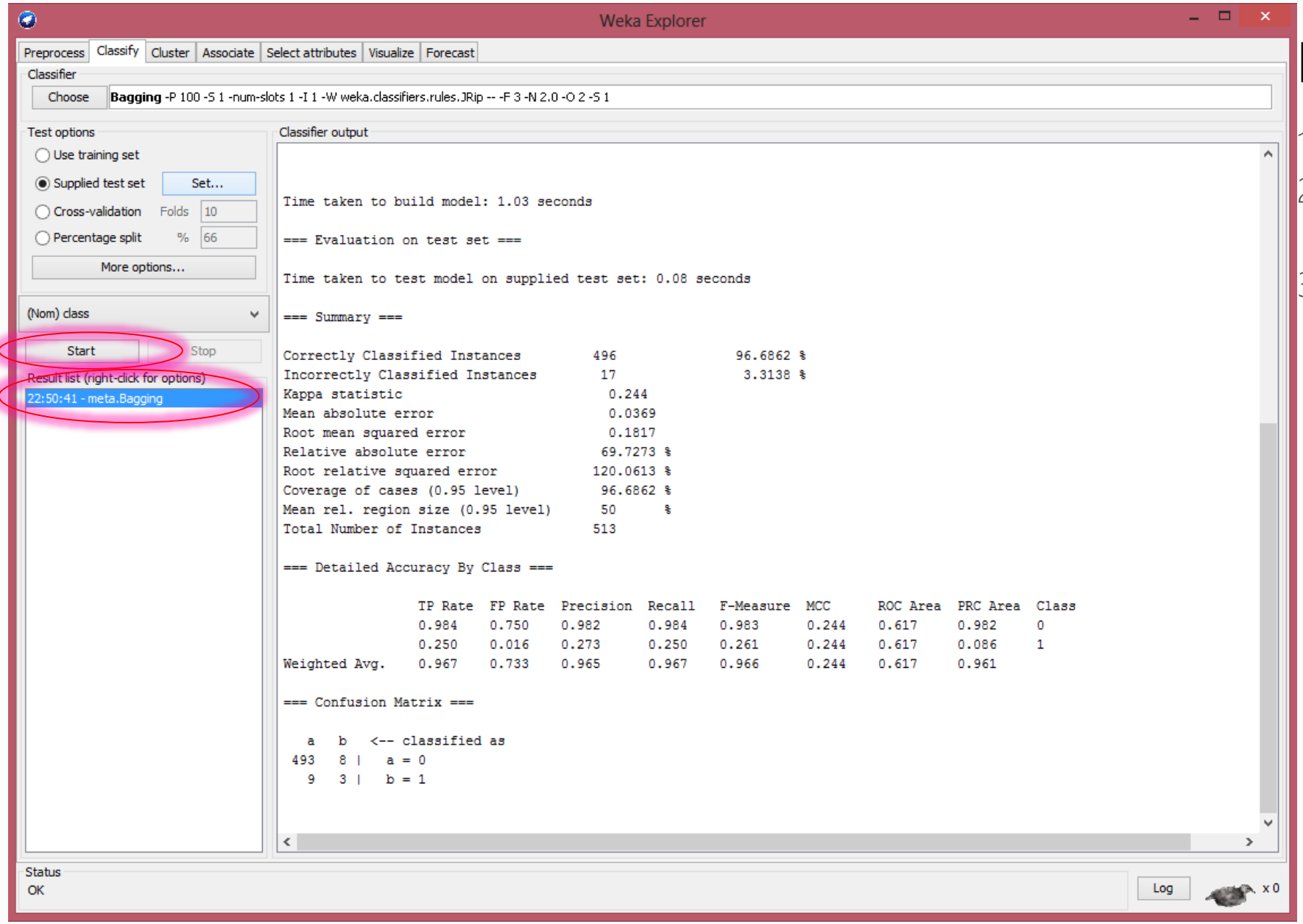
Size of the bag relative to full train set

Classifier to use

Number of models/bagging to build

Seed for random selection of tuples

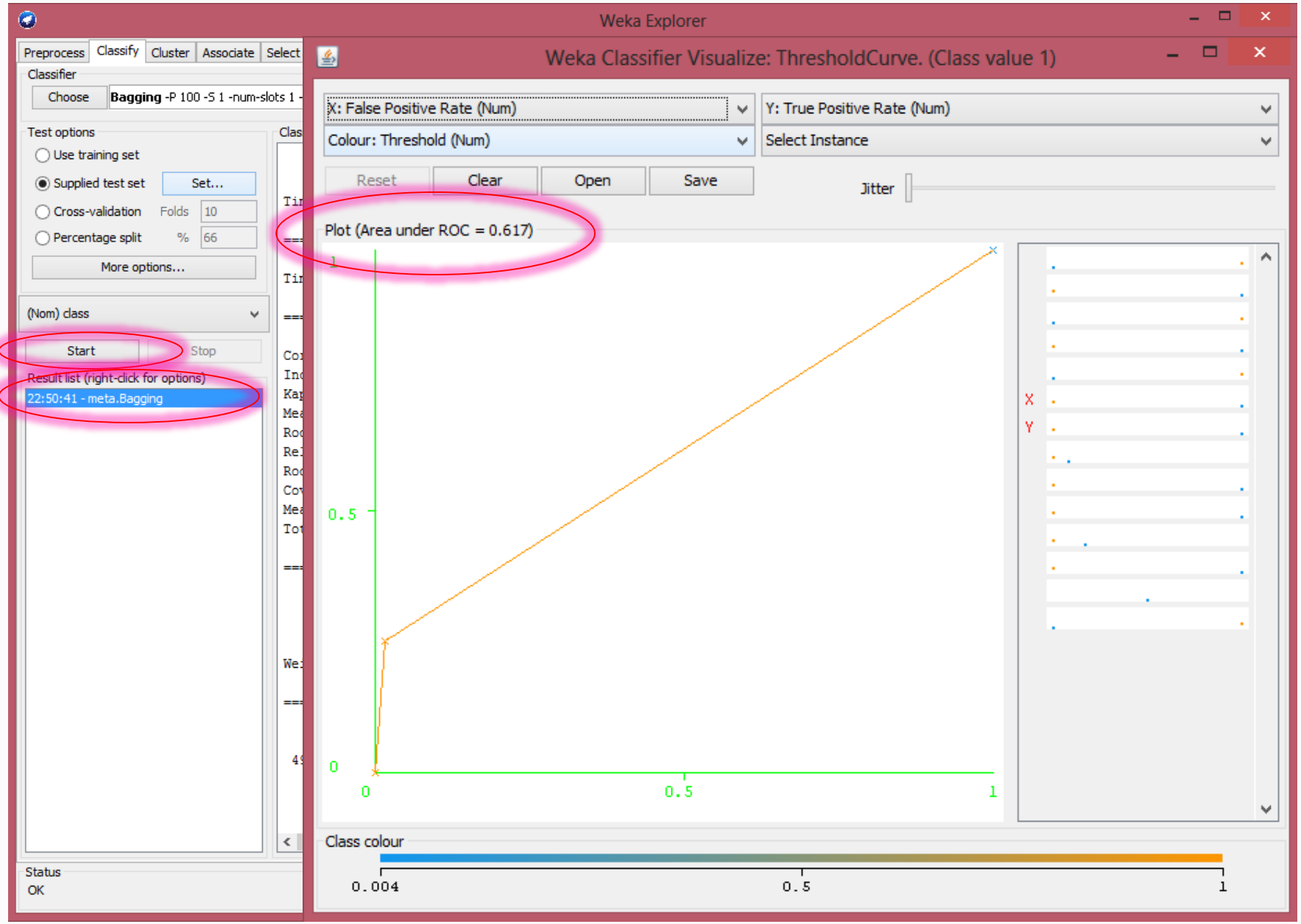
Bagging



Prepare one model

1. Click Start to run model
2. Right-click on the line last line of the Result list
3. Select Visualize threshold curve and then 1 (we want to see perf of True Positives "1")

Bagging

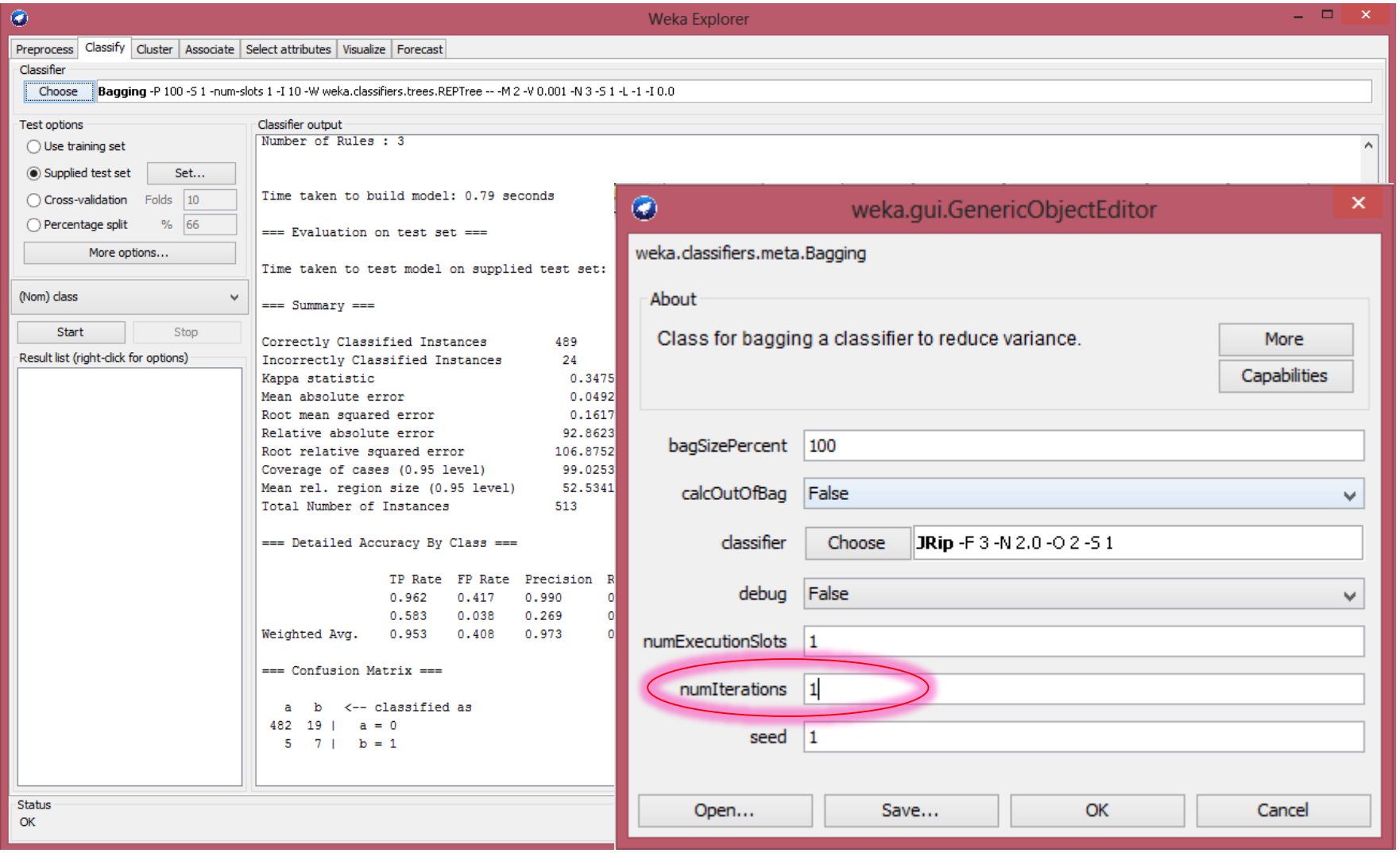


Prepare one model

1. Click Start to run model
2. Right-click on the line last line of the Result list
3. Select Visualize threshold curve and then 1.

4. ROC AUC value (about 0.6) is rather poor which means that a large portion of active compounds cannot be retrieved using only one rule set.

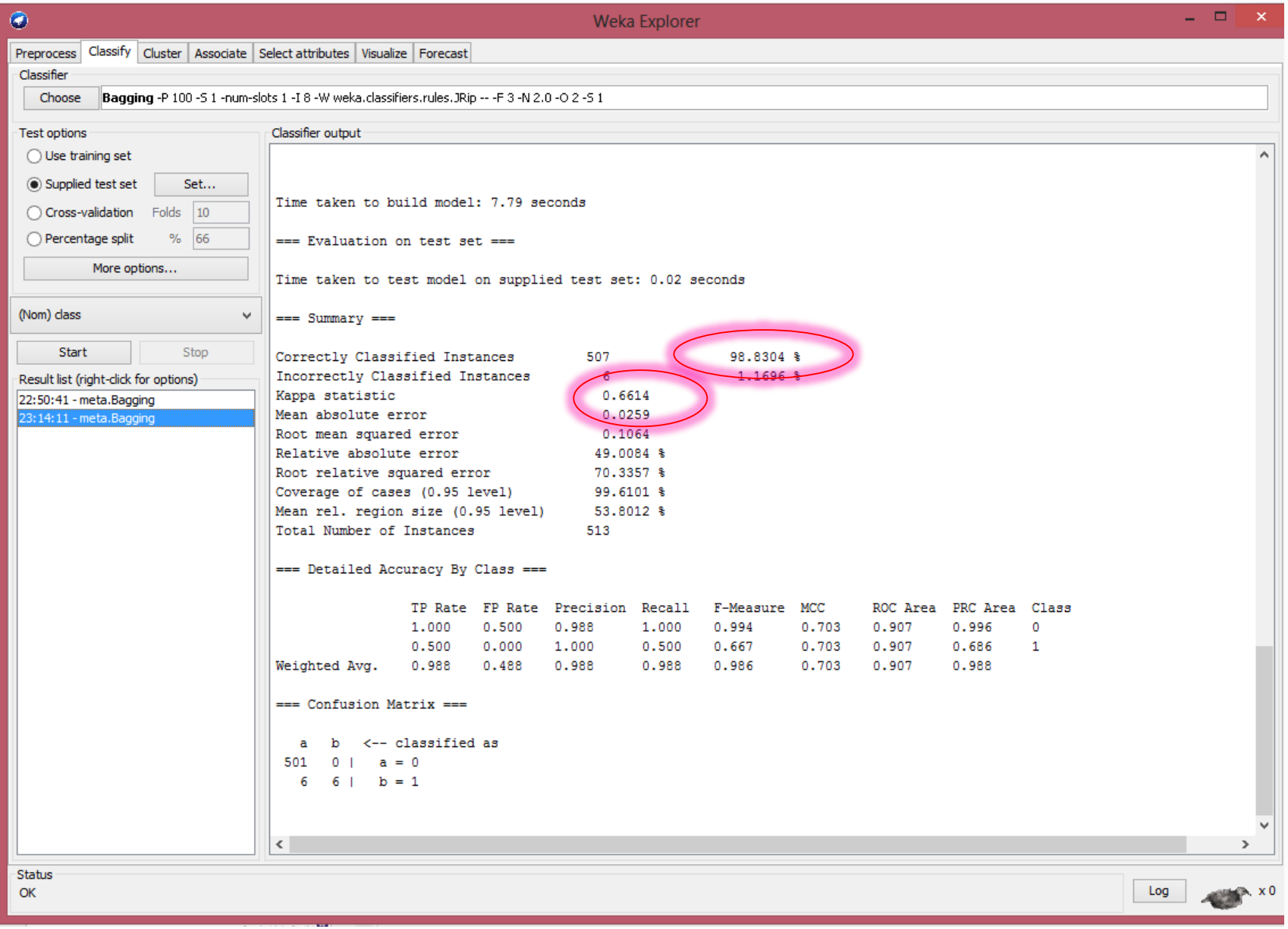
Bagging



Prepare one model

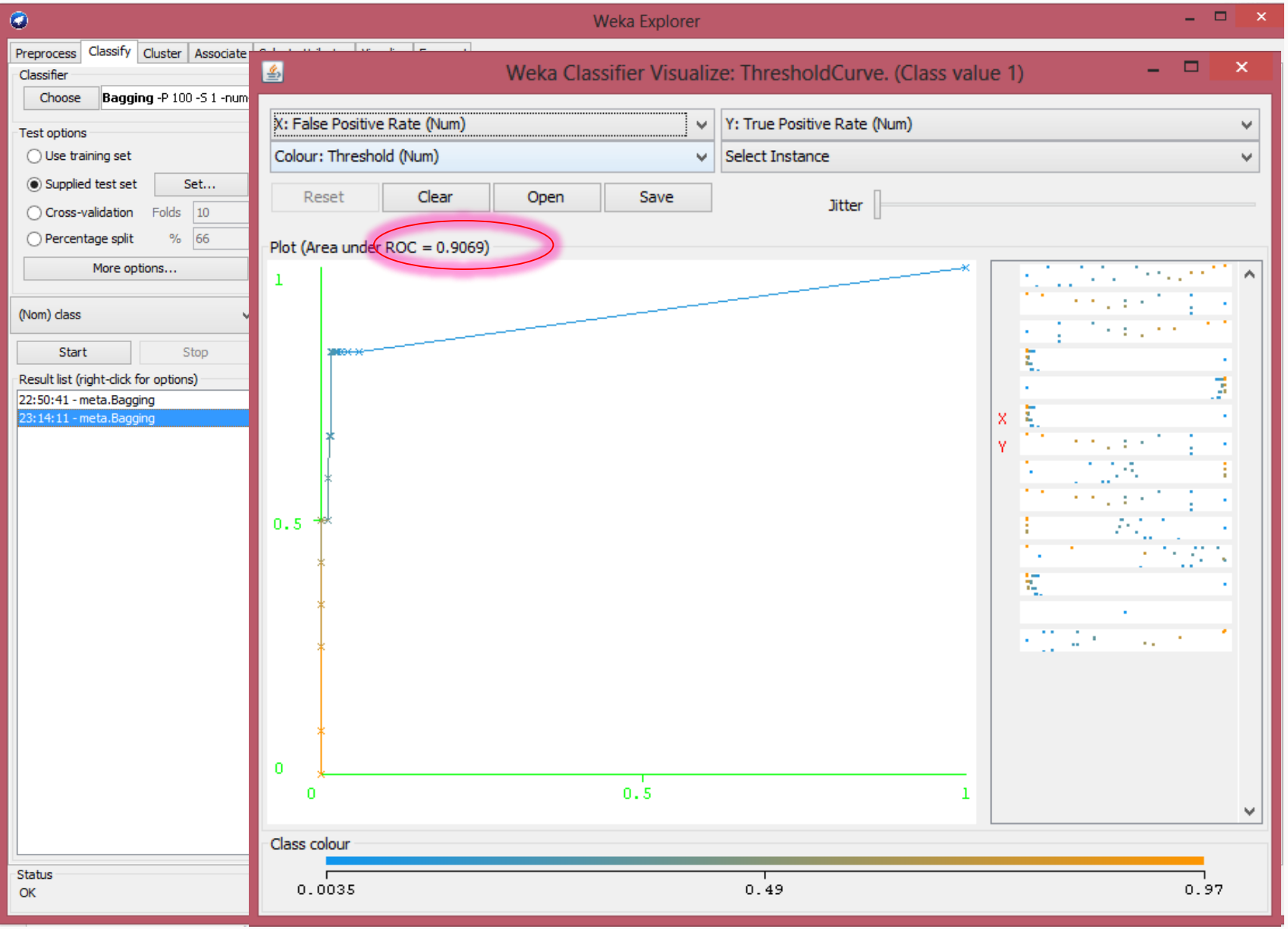
1. Click **Classify**, then **Choose**.
2. Select **classifiers->meta->Bagging**.
3. Click on the name of the method to the right of the Choose button.
4. **Choose** then select **classifiers->rules->JRip**.

5. Try this our with **numIterations** from **3 … 8**



Runs 3...8

1. Click Start to run model
2. Right-click on the line last line of the Result list
3. Select Visualize threshold curve and then 1 (we want to see perf of True Positives "1")



Runs 3...8

1. Click Start to run model
2. Right-click on the line last line of the Result list
3. Select Visualize threshold curve and then 1 (we want to see perf of True Positives "1")

4. One can see that ROC AUC for the consensus model increases up to 0.9.

AdaBoost

Generate a sequence of base-learners each focusing on previous one's errors

Training:

For all $\{x^t, r^t\}_{t=1}^N \in \mathcal{X}$, initialize $p_1^t = 1/N$

For all base-learners $j = 1, \ldots, L$

Randomly draw $\mathcal{X}_j$ from $\mathcal{X}$ with probabilities p_j^t

Train d_j using $\mathcal{X}_j$

For each (x^t, r^t), calculate $y_j^t \leftarrow d_j(x^t)$

Calculate error rate: $\epsilon_j \leftarrow \sum_t p_j^t \cdot 1(y_j^t \neq r^t)$

If $\epsilon_j > 1/2$, then $L \leftarrow j - 1$; stop

$\beta_j \leftarrow \epsilon_j / (1 - \epsilon_j)$

For each (x^t, r^t), decrease probabilities if correct:

If $y_j^t = r^t$ $p_{j+1}^t \leftarrow \beta_j p_j^t$ Else $p_{j+1}^t \leftarrow p_j^t$

Normalize probabilities:

$$Z_j \leftarrow \sum_t p_{j+1}^t; \quad p_{j+1}^t \leftarrow p_{j+1}^t / Z_j$$

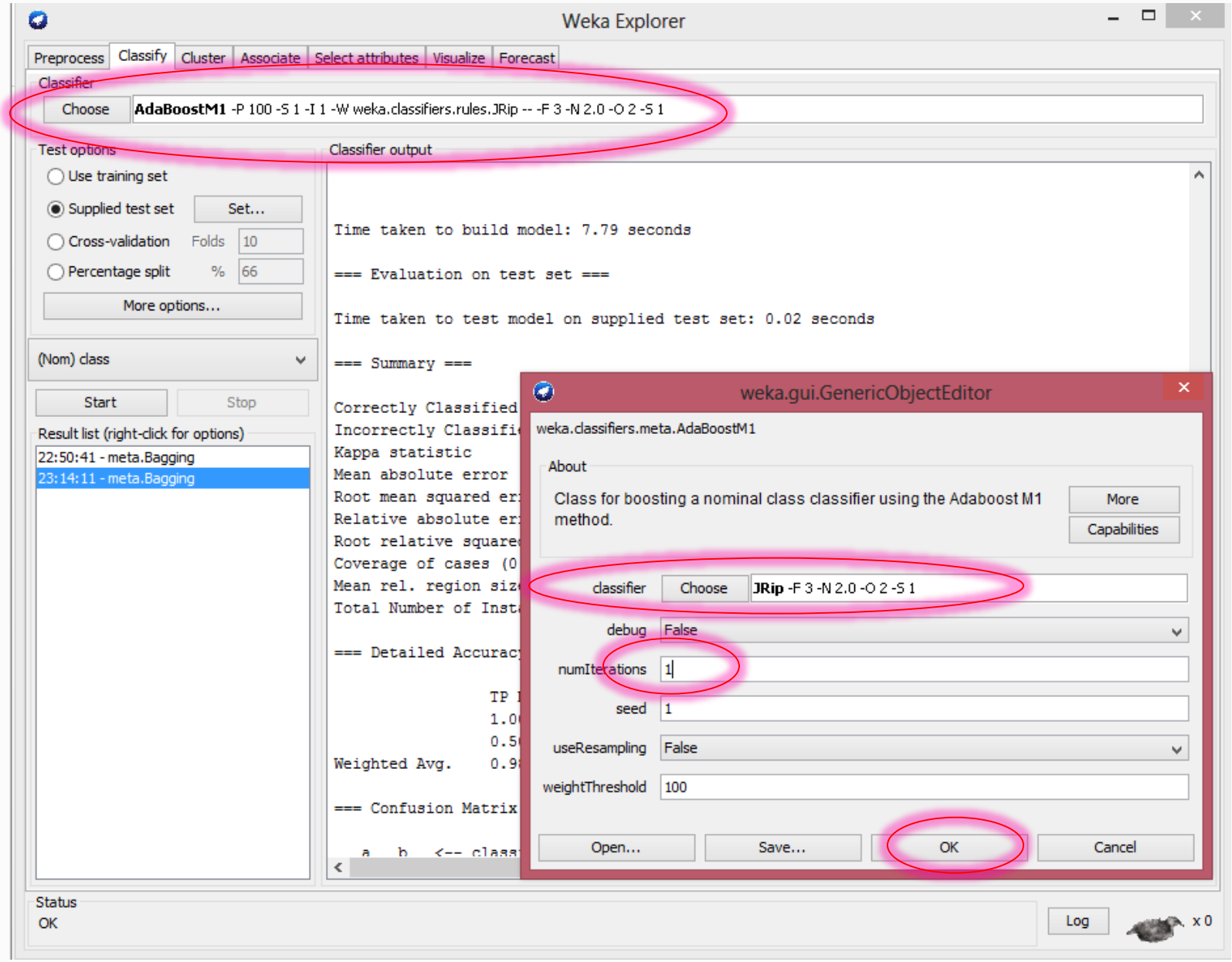
Testing:

Given x, calculate $d_j(x), j = 1, \ldots, L$

Calculate class outputs, $i = 1, \ldots, K$:

$$y_i = \sum_{j=1}^{L} \left(\log \frac{1}{\beta_j} \right) d_{ji}(x)$$

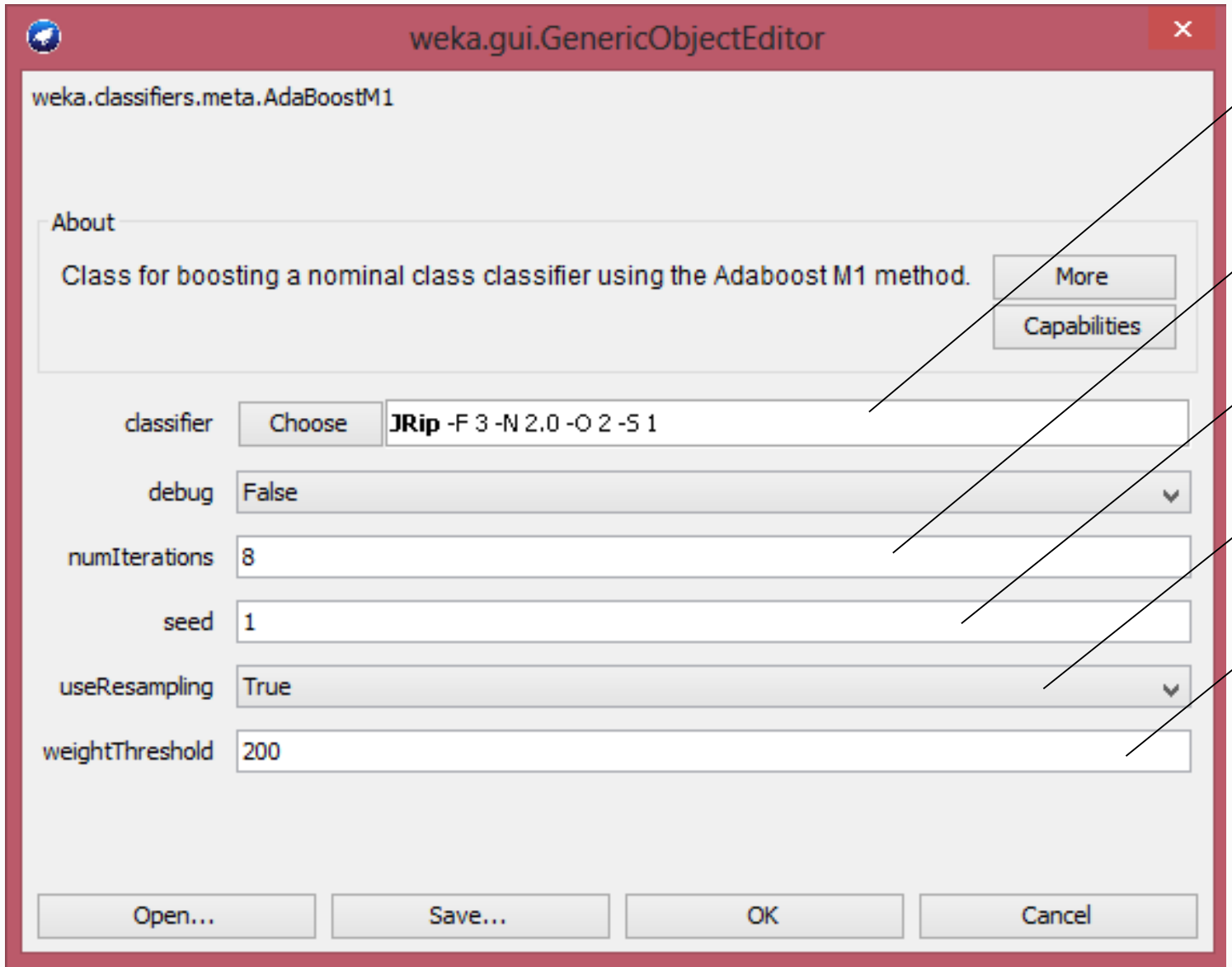
Boosting



Prepare one model

1. Click on the **Classify** tab.
2. Click **Choose** and select the method **classifiers->meta->AdaBoostM1**.
3. Click **AdaBoostM1** in the box to the right of the button. The configuration interface of the method appears.
4. Click **Choose** of this interface and select the method **classifiers->rules>JRip**.
5. Set the **numIterations** to 1.
6. Click on the button **OK**.

Quick double-click on boosting…



Classifier to use…

Number of models to build

Seed for random selection of tuples

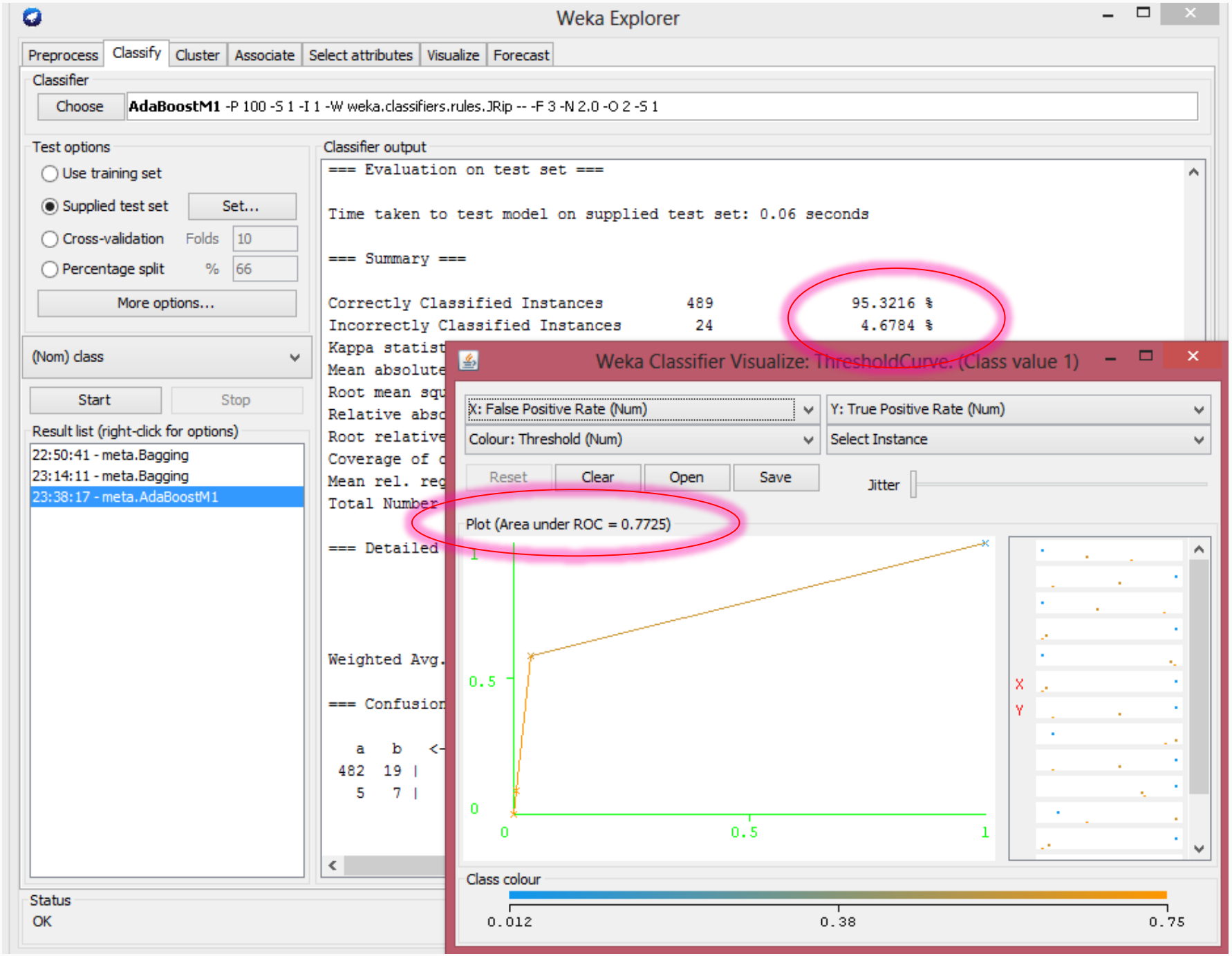
Resample or reweighting

Threshold for weight pruning

Boosting, 1 Model

1. Click **Start** to run model
2. Right-click on the line last line of the Result list
3. Select **Visualize** threshold curve and then 1.

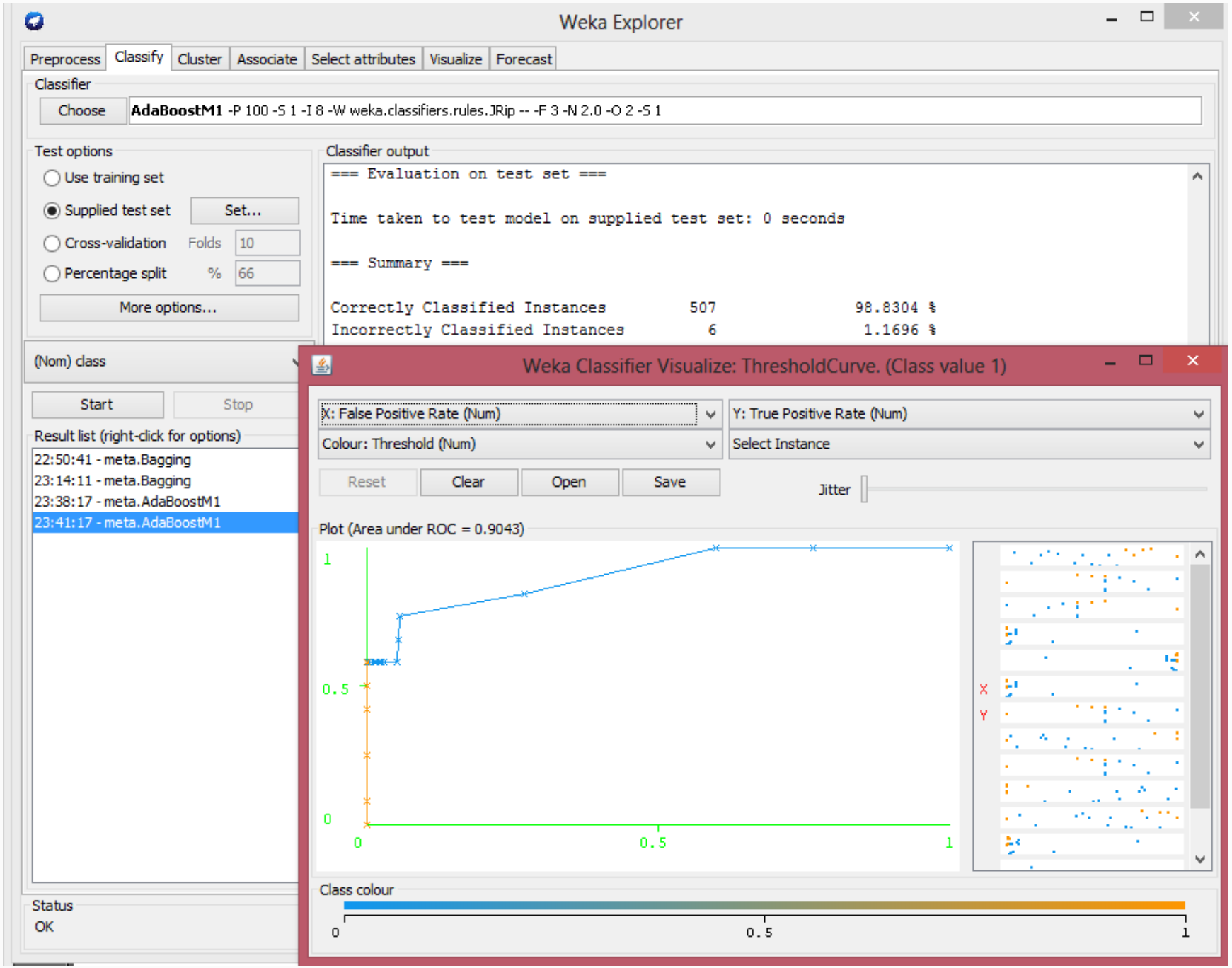
4. ROC AUC value (about 0.7) is higher than simple bagging with one model…



Boosting Runs 3…8

1. Click **Start** to run model
2. Right-click on the line last line of the Result list
3. Select **Visualize** threshold curve and then **1** (we want to see perf of True Positives "1")

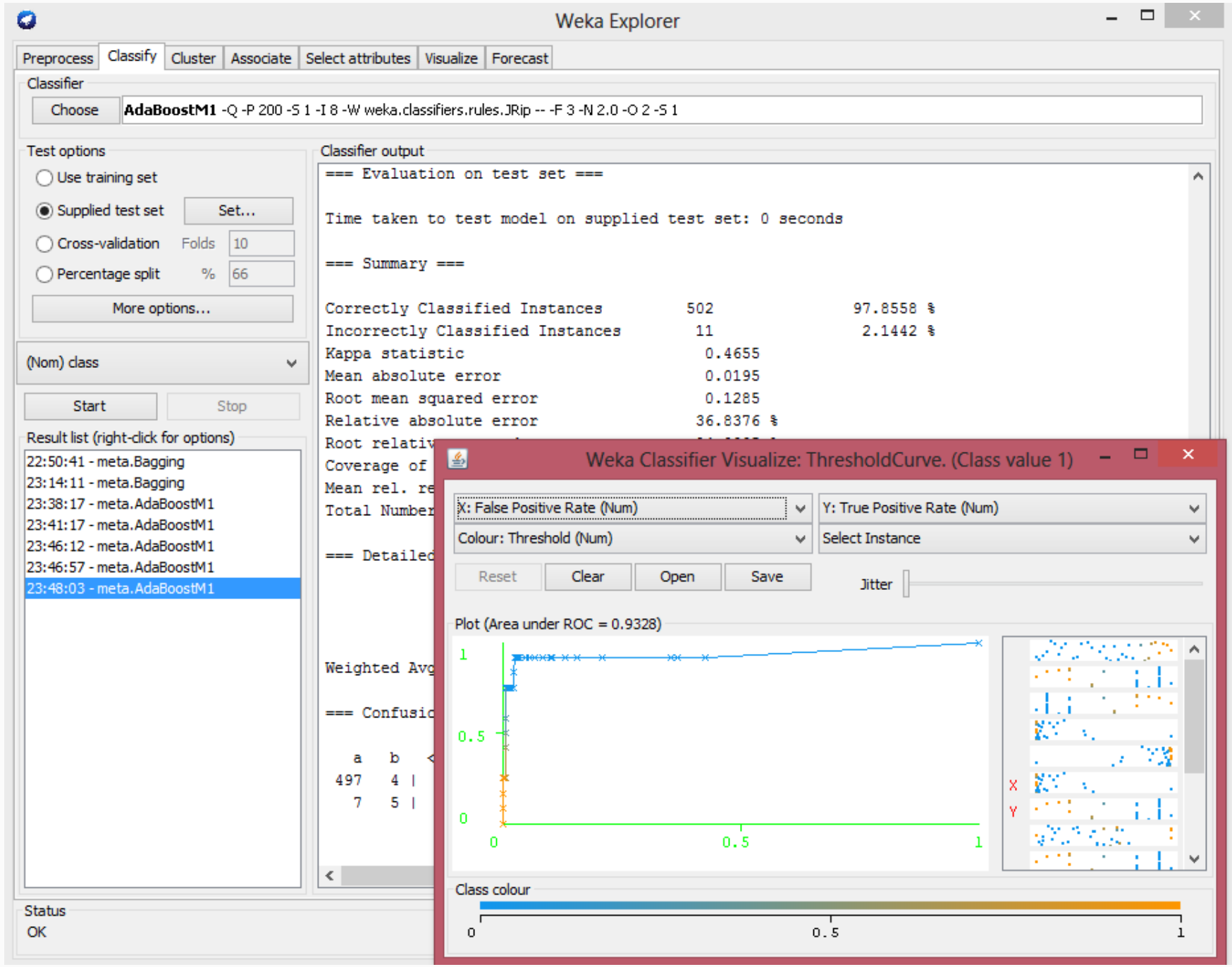
4. One can see that ROC AUC for boosting increases to 0.9 and increases faster than bagging..



Boosting Runs 8

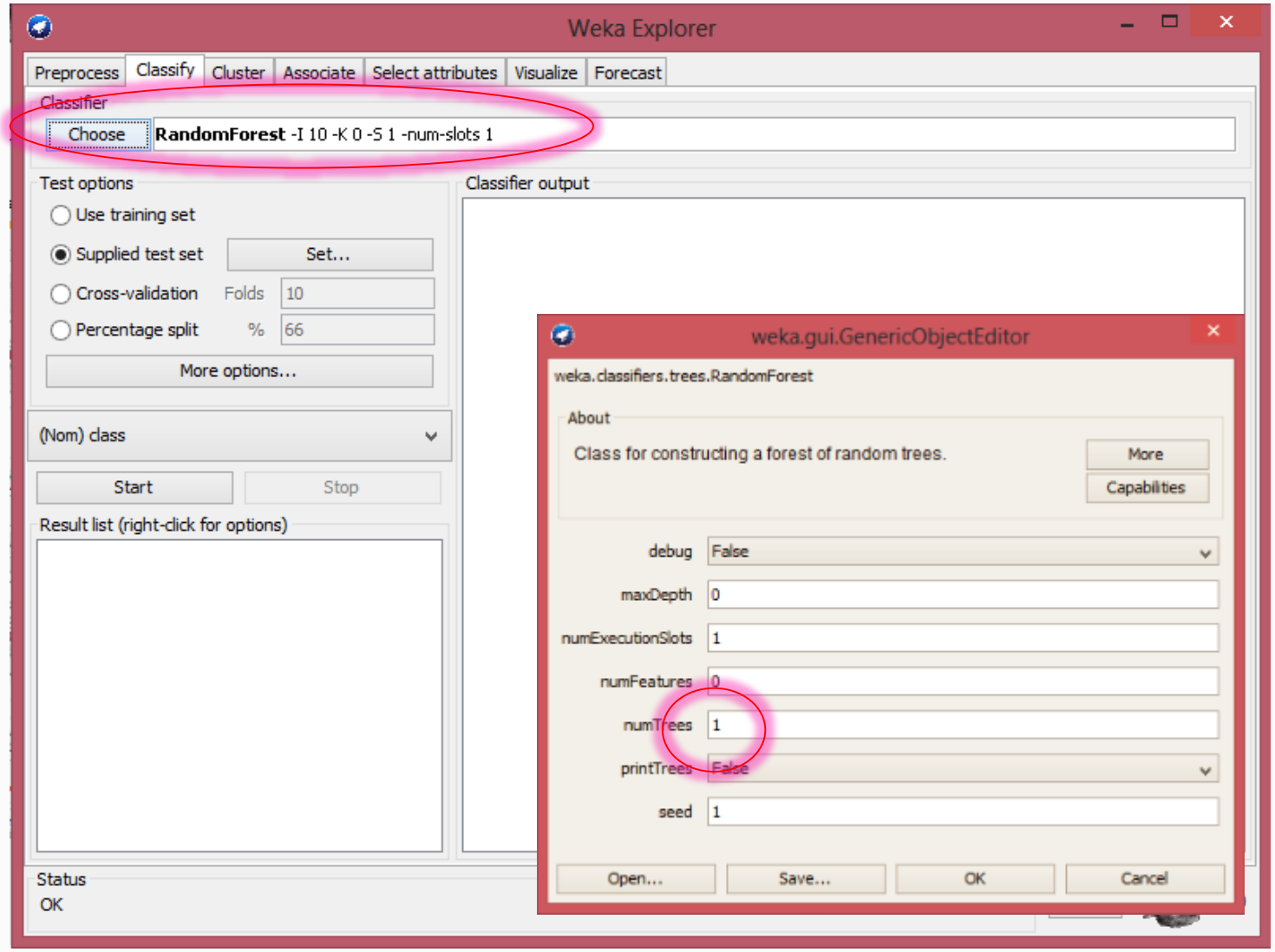
1. Set Resampling to TRUE

2. ROC AUC increases to 0.93 for True Positives…



Random Forest

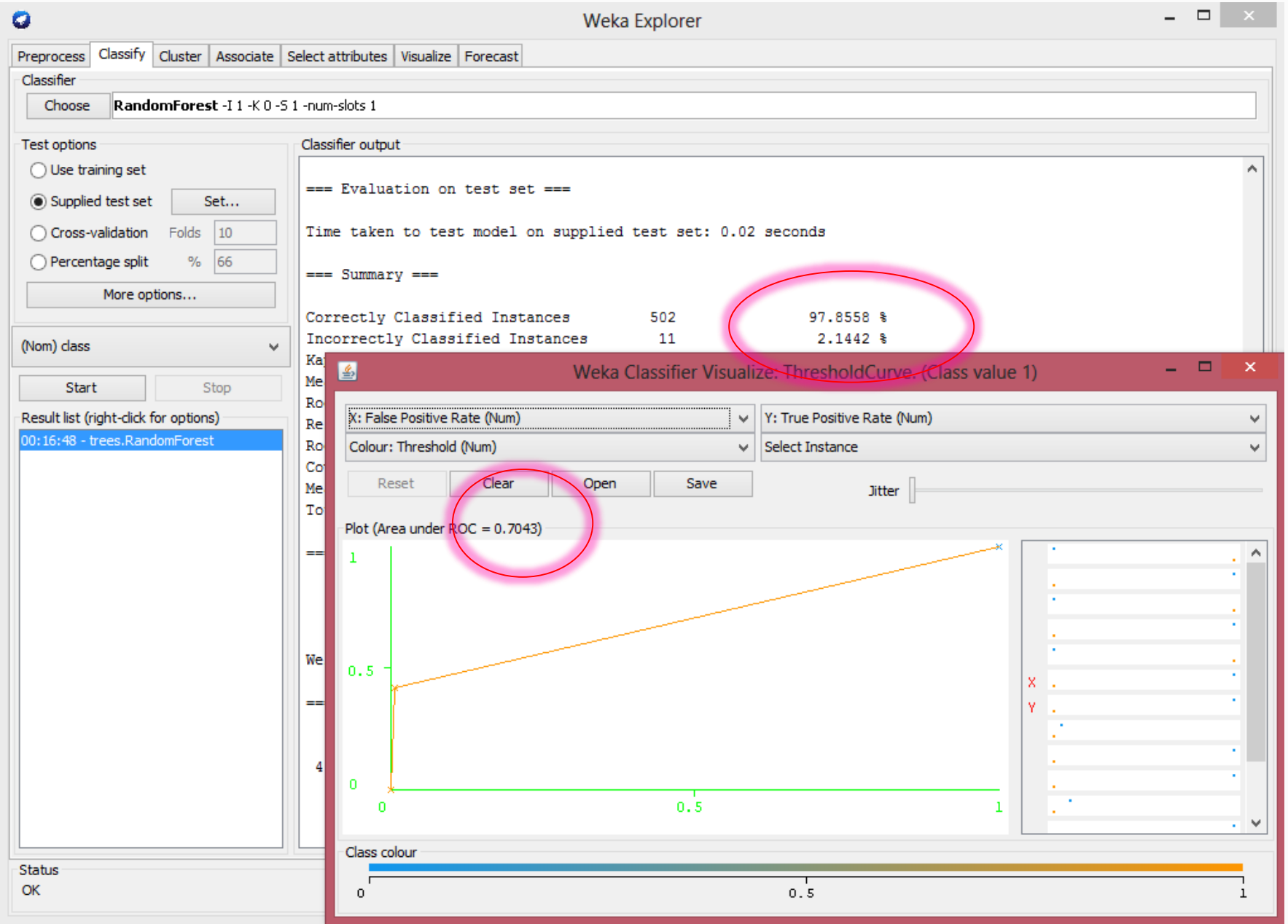
The Random Forest method is based on bagging models built using the Random Tree method, in which classification trees are grown on a random subset of descriptors



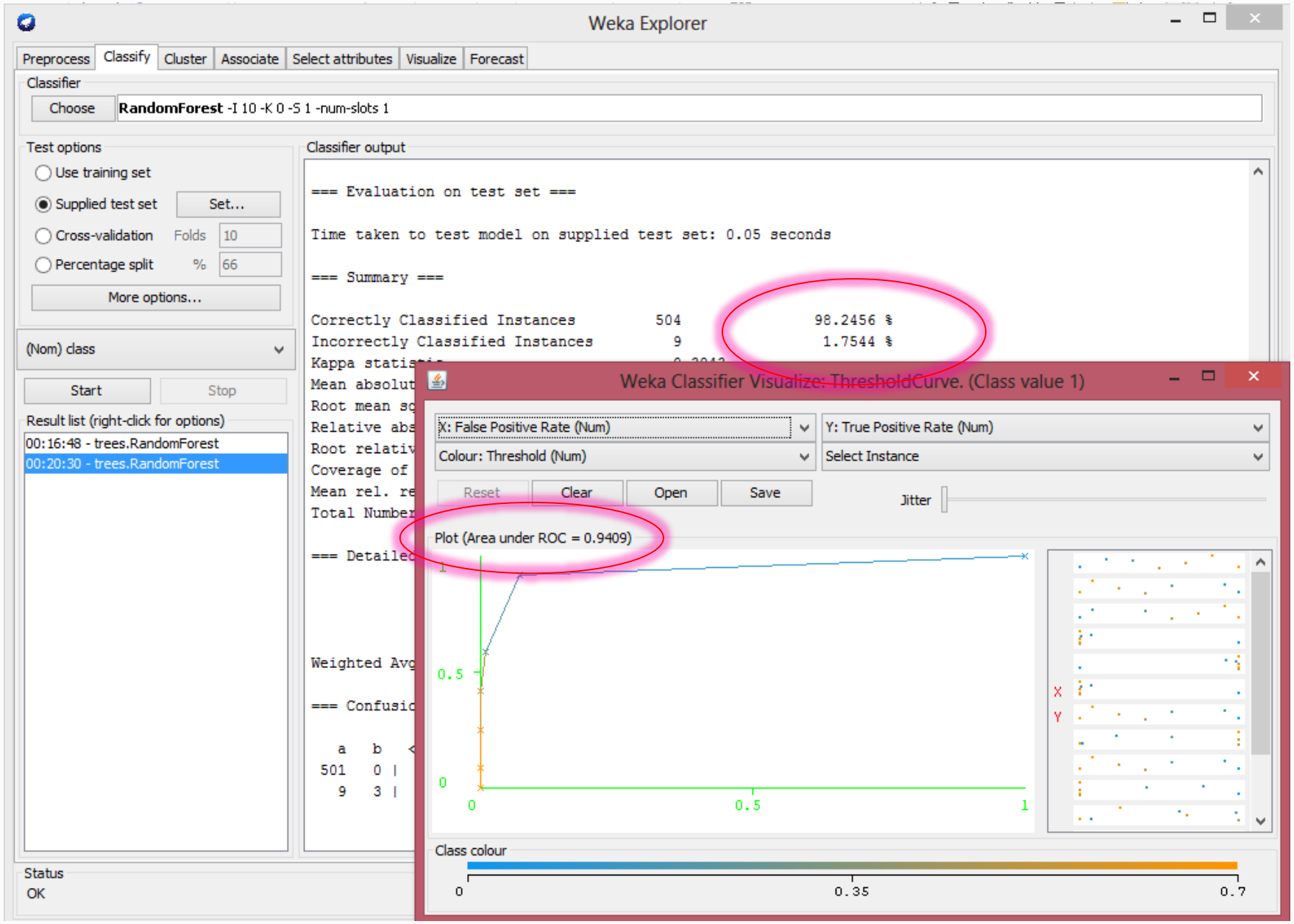
Random Forest

1. Click on **Classify** tab. Make sure that the test set is supplied.
2. Click **Choose** and select the method **classifiers->tree->RandomForest**.
3. Click on **RandomForest**, a configuration interface appears.
4. Set the **numTrees** to **1**
5. Click the button **OK**.
6. Click **Start**.

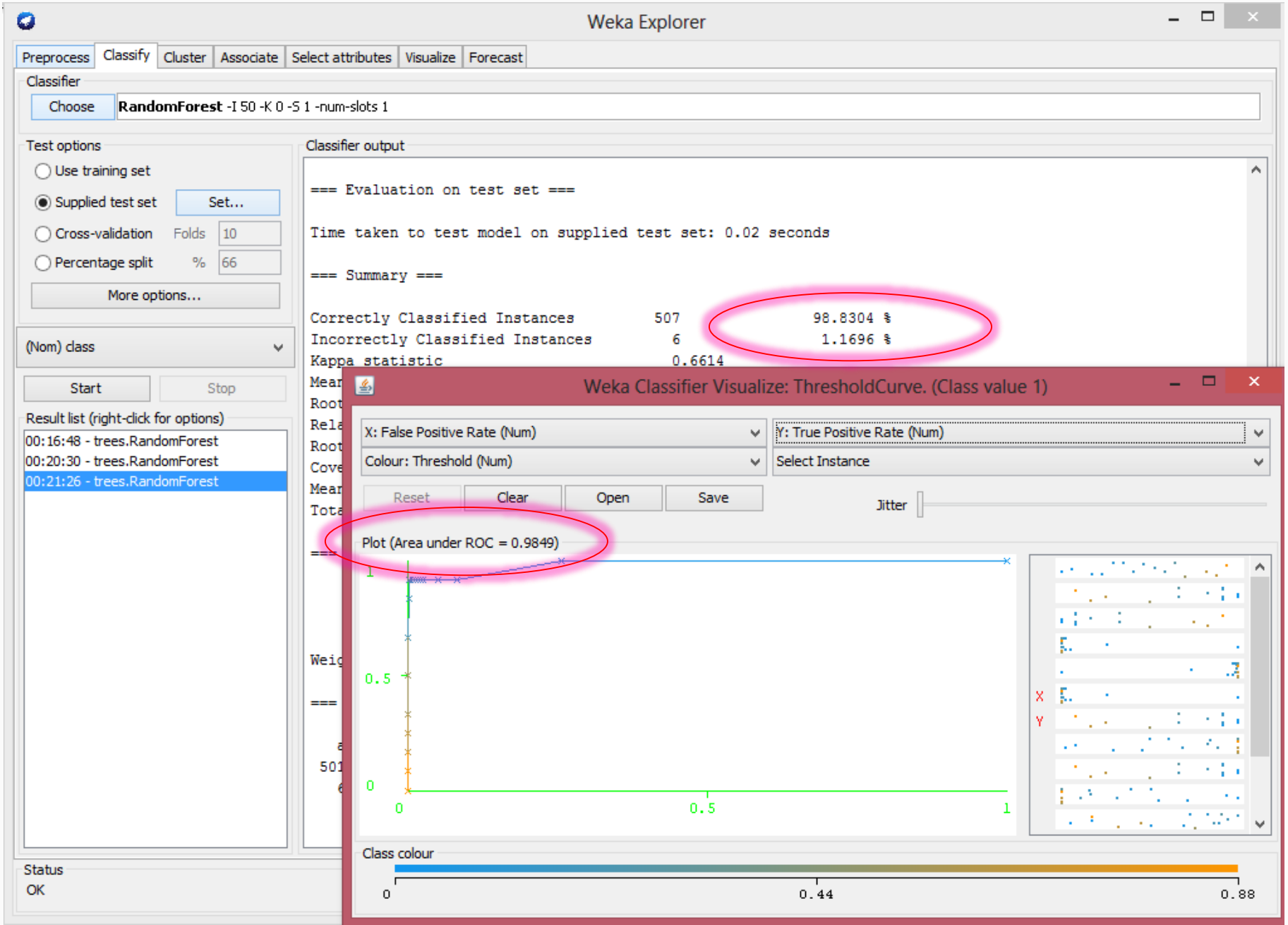
Random Forest, 1 tree..



Random Forest, 10 trees…



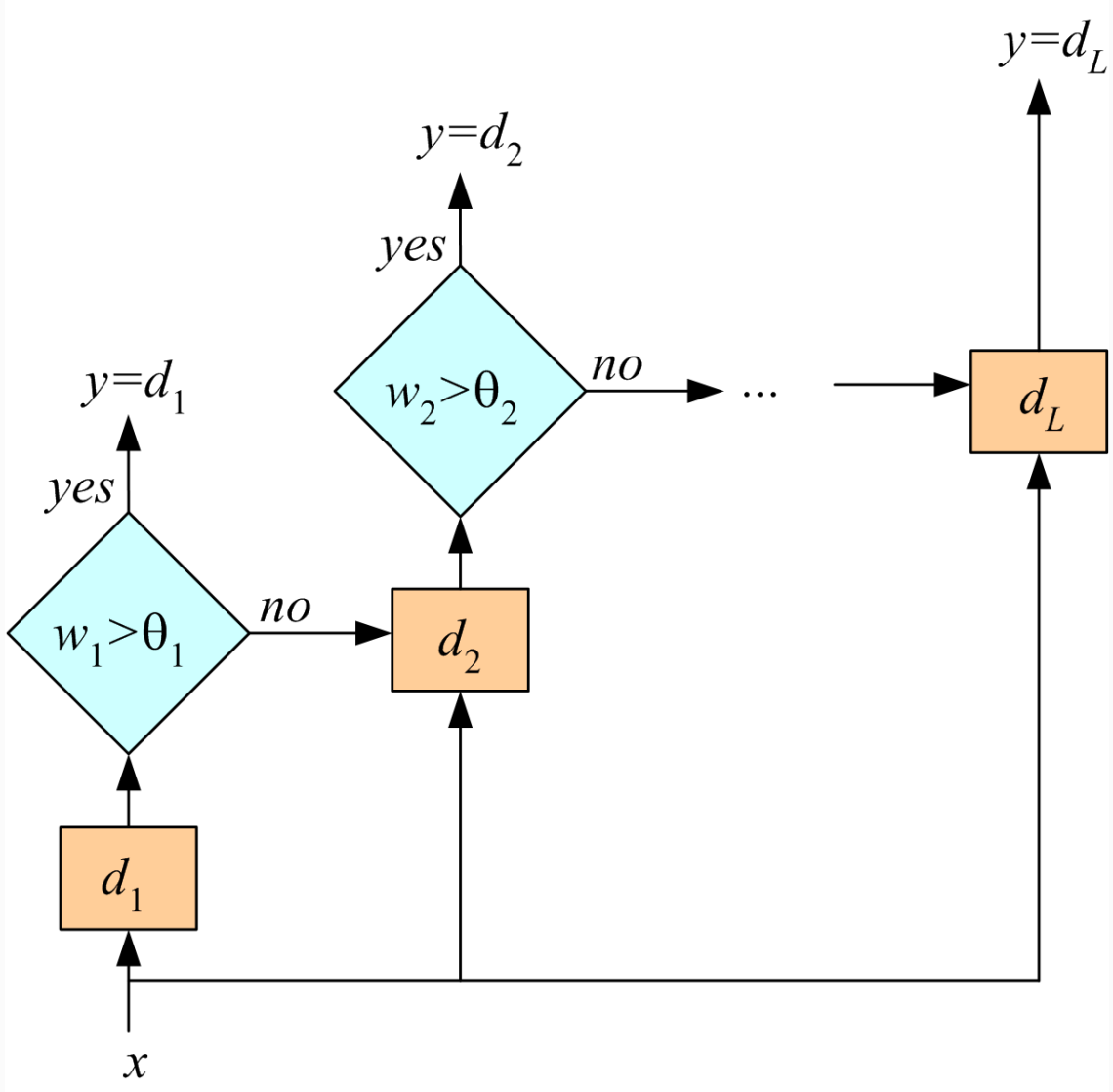
Random Forest, 50 trees…



Stacking

Stacking involves *training a learning algorithm* to combine the predictions of several learning algorithms. First, all the algorithms are trained using the available data, then a *combiner algorithm is trained* to make a final prediction using all the predictions of the other algorithms as additional inputs (StackingC uses their PDFs as well). If an arbitrary combiner algorithm is used, stacking can theoretically represent any of the ensemble techniques, including gating.

The two top-performers in the Netflix competition utilized blending, which may be considered to be a form of stacking



Course Project…

Data Science

Deriving Knowledge from Data at Scale

That's all for tonight….

Nov 30th, 2015