**CSC 722 AML – Home work 4**

1. **(35 pts.) From among the students presentations so far, select a paper you did not present yourself and did not choose for Assignment 2 or 3. Decribe and critically discuss this paper. What is the paper about? What problem is it addressing? What approach does it use? What are the advantages and diadvantages of this approach? How were the results presented? How well does the paper describe what the authors did? What is your overall assessment of this paper, good and bad?**

The paper I selected is “Sampling from Large Graphs “

Authors are : Jure Leskovec, Christos Faloustsos

This paper was presented by Anshul Atriek

***Introduction and Discussion :***

Large graphs in this data era are becoming very common. Anything that can be represented as a network would have a large graph.

EX : citation graphs, Facebook posts/friends

To understand these graphs for analyzing them for future predictions, we can use certain graph algorithms. This paper’s work is on these important graph algorithms. The main focus is on how to analyze and visualize large graphs and compress the size of the graph to smaller sizes so that analysis can be faster.

***Motivation and Problem Statement:***

The paper mainly has 3 Research questions :

* What sampling methods to use? What is a good sampling method? What’s the best?
* What is a good sample size? How small can we really go?
* How do we measure the goodness of a sample and/or a sampling method?

***Previous Related Work and it’s limitation:***

* Sampling on graphs has been used in many different ways
* Internet modeling community studied sampling from undirected graphs and concluded that some graph properties can be preserved by random-node selection with sample sizes down to 30%.
* Works on graph compression focused on transforming the graph to speed up algorithms

**Limitation :**

* A large set of graph properties weren’t able to be matched.
* Sampling was done just to visualize the graph but the essential features were not tried to be retained.

***Experiments (Approach)***

The sampling of large graphs was done by considering 2 different approaches : Scale down and Back in time.

**Evaluation for Static graphs :**

• S1: In-degree distribution: for every degree d, we count the number of nodes with in-degree d. Typically it follows a power-law and some other heavy tailed distribution

• S2: Out-degree distribution.

• S3: The distribution of sizes of weakly connected components (“wcc”): a set of nodes is weakly connected if for any pair of nodes u and v there exists an undirected path from u to v.

• S4: The distribution of sizes of strongly connected components (“scc”): a set of nodes is strongly connected, if for any pair of nodes u and v, there exists a directed path from u to v and from v to u.

• S5: Hop-plot: the number P(h) of reachable pairs of nodes at distance h or less; h is the number of hops

• S6: Hop-plot on the largest WCC.

• S7: The distribution of the first left singular vector of the graph adjacency matrix versus the rank.

• S8: The distribution of singular values of the graph adjacency matrix versus the rank. Spectral properties of graphs often follow a heavy-tailed distribution

• S9: The distribution of the clustering coefficient Cd defined as follows.

**Evaluation for temporal graphs :**

• T1: Densification Power Law (DPL)

• T2: The effective diameter of the graph over time, which is defined as the minimum number of hops in which 90% of all connected pairs of nodes can reach each other

• T3: The normalized size of the largest connected component (CC) over time.

• T4: The largest singular value of graph adjacency matrix over time.

• T5: Average clustering coefficient C over time [16]: C at time t is the average Cv of all nodes v in graph at time t.

* Sampling was done using 3 techniques :
* They are :
  + Sampling Random Node Selection : Further divided into

Random Node (RN)

Random PageRank Node (RPN)

Random Degree Node (RDN)

* + Sampling by Random Edge Selection :

Random Edge (RE)

Random Node-Edge (RNE)

Hybrid (HYB)

* + Sampling by exploration

Random Node Neighbor (RNN)

Random Walk (RW)

Random Jump (RJ)

Forest Fire (FF)

Five datasets were used :

One static graph

four graphs with temporal information

Both the approaches Scale down and back in time were used over the 5 datasets.

In **Scale Down** , the goal was to create a sample graph S, that will match the properties of the target graph G.

**Step 1: Matching the graph patterns**

Three clusters of algorithms were formed.

[RDN, RJ ,RW] [FFF, RPN, RN], [RE, RNE, HYB]

**Step 2: Evaluation using the D-statistic**

Evaluating the sampling algorithms against all 14 graph static and temporal patterns.

For Scale-down sampling criteria the temporal patterns are essentially flat.

Distribution of weakly connected components is best matched by edge selection techniques

In **Back in time**, goal is to match the graph as it grows and evolves.

**Step 1: Matching the graph patterns**

These 3 clusters were formed

[RW, RJ, RDN], [FF, RN, RPN], [RW, RJ,RDN]

**Step 2: Evaluation using the D-statistic**

FF performed best -average D-statistic 0.13

Second group was then formed by RN, RW with D-stat of 0.16

***Advantages of this approach:***

The usage of empirical rules was very effective.

The list of properties was pretty big.

The sampling tries to match graph against the 14 (9+5)

***Disadvantages of this approach :***

Generalization was done on only 2 sampling methods.

The Back in time sampling method didn’t require extensive experimentation – how good is this in future.

***Results:***

**Scale Down :**

* The temporal patterns are essentially flat, since regardless of a sample size we want to match the property of the final graph.
* The sample size is small, so the RN and RNN are far off. Edge selection techniques (RE, RNE and HYB) give too sparse samples which contain no triangles. FF works fine.

**Back In Time :**

* Edge selection techniques produce disconnected graphs and thus underestimate the clustering.
* FF, RN and RPN match the size of connected component and clustering over time.
* Overall, FF performed best
* Second group was then formed by RN, RW .
* Again, edge selection performed worst.
* Back-in-time goal, we obtain good results for 0 ≤ pf ≤ 0.4
* Scale-down goal best performance was obtained with high values of pf (pf ≥ 0.6)

***Clarity given on the work by authors:***

Although the authors explained in a very detailed way everything, there are few points which are not covered like :

Only 2 ways to look at graph sampling were explained. Are there only these 2 ways? If not, why weren’t other ways used?

In other Sampling techniques, it was said that BFS and DFS did not perform well – no reasoning was given for this.

***Assessment:***

Overall the paper was very well explained and mainly the examples which were mentioned were very meaningful. The authors presented this paper in the year 2006. At that time (almost a decade back), large graphs were not as popular as they are now. Yet they did a good job in it because Sampling is very important when the data is huge

1. **(35 pts.) Compare and contrast Adaboost and Bagging in**[**Weka**](http://www.cs.waikato.ac.nz/ml/weka/)**, using one of the two standard benchmark datasets from the UCI repository**
   * [**Zoo Animals**](http://archive.ics.uci.edu/ml/datasets/Zoo)
   * [**Wisconsin Breast Cancer**](http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29)

**as training data.**

**In your comparison, use the same classifier (or Base Classifier in Adaboost) for each method. Include at least one decision tree method. (The standard decision tree method in Weka is called J48.)**

**Depending on the method, experiment with different (base) learners, number of iterations, bag sizes, and possibly other parameter settings.**

**In Weka Explorer, clicking the field next to the Choose button will pull down a menu that lets you change things.**

**Try to determine the best choice for everything. Explain what you did, which settings appear to work best, and your best guess as to why.**

Solution :

Using Zoo Animals dataset:

The following experiments were done :

NOTE :

* To maintain consistency among the experiments, cross validation of 10 was used for all experiments.
* Batch size was set to 100 for all experiments
* Only the screenshot of the models with highest and least accuracy is attached in the report, the others will be in the folder ‘models’

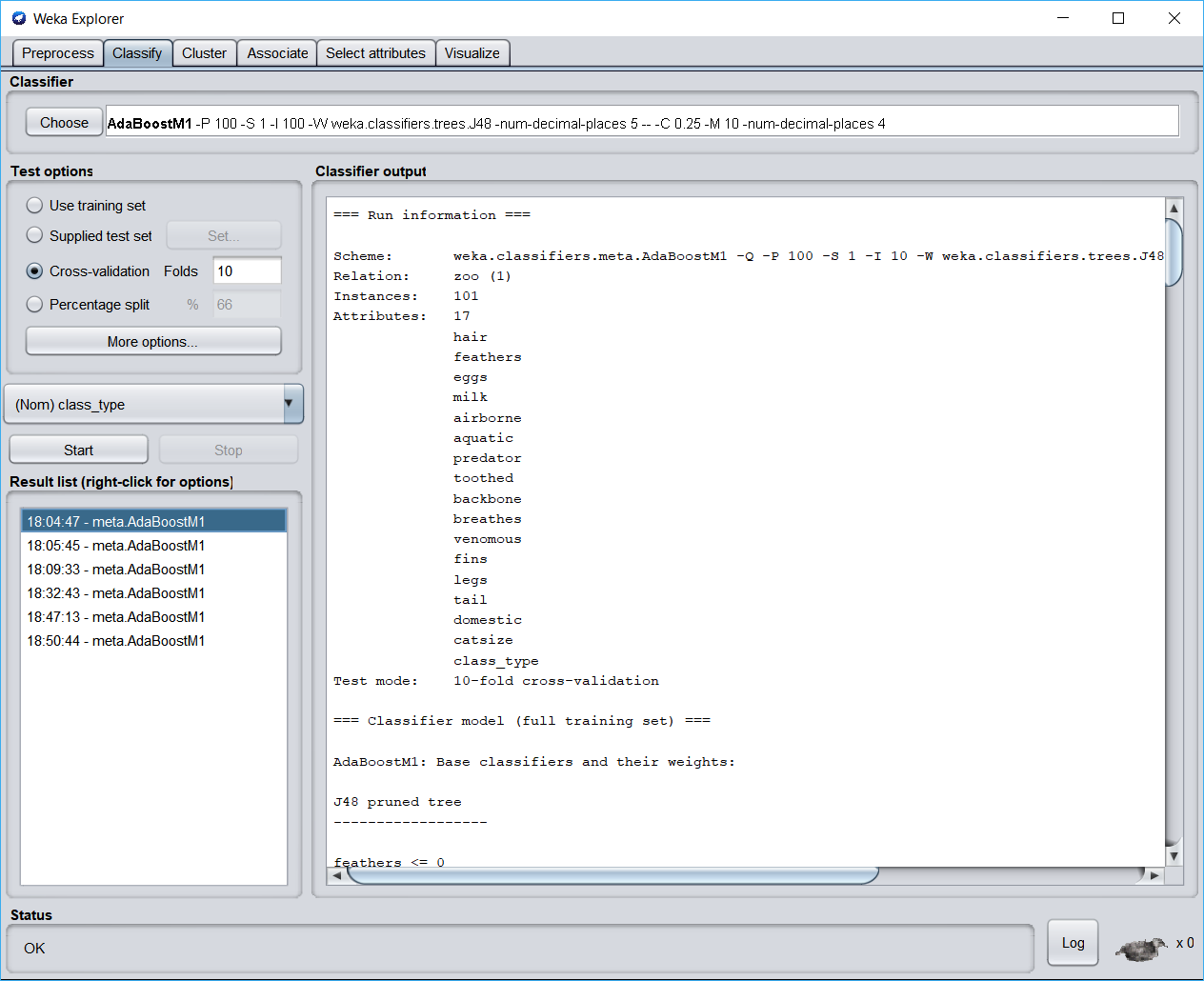
1. J48 on AdaBoost

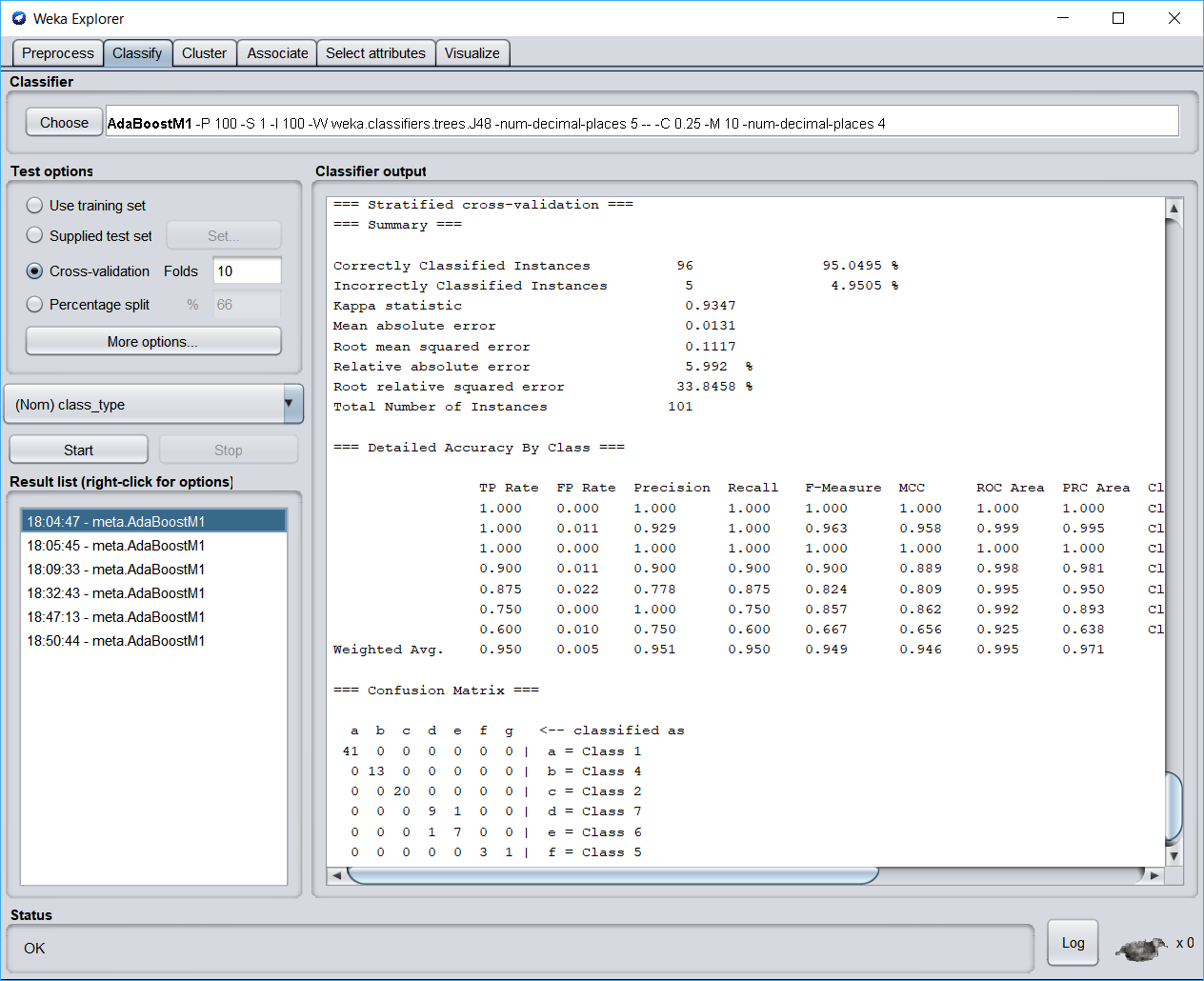
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Adaboost Specific parameters | | | J48 specific parameters | | | Evaluation Metrics | | |
| Capabilities | NumIterations | UseResampling | ReducedErrorPruning | NumFolds | MinNumObj | Accuracy | RMSE | |
| true | 10 | true | True | 3 | 2 | 95.04 | 0.11 |
| True | 30 | true | True | 5 | 5 | 94.05 | 0.12 |
| False | 5 | false | False | 3 | 10 | 87.12 | 0.15 |
| True | 200 | true | true | 6 | 15 | 80.19 | 0.22 |
| False | 100 | false | False | 3 | 10 | 89.10 | 0.14 |
| False | 200 | false | False | 3 | 10 | 89.10 | 0.14 |
|  |  |  |  |  |  |  |  |

Observations :

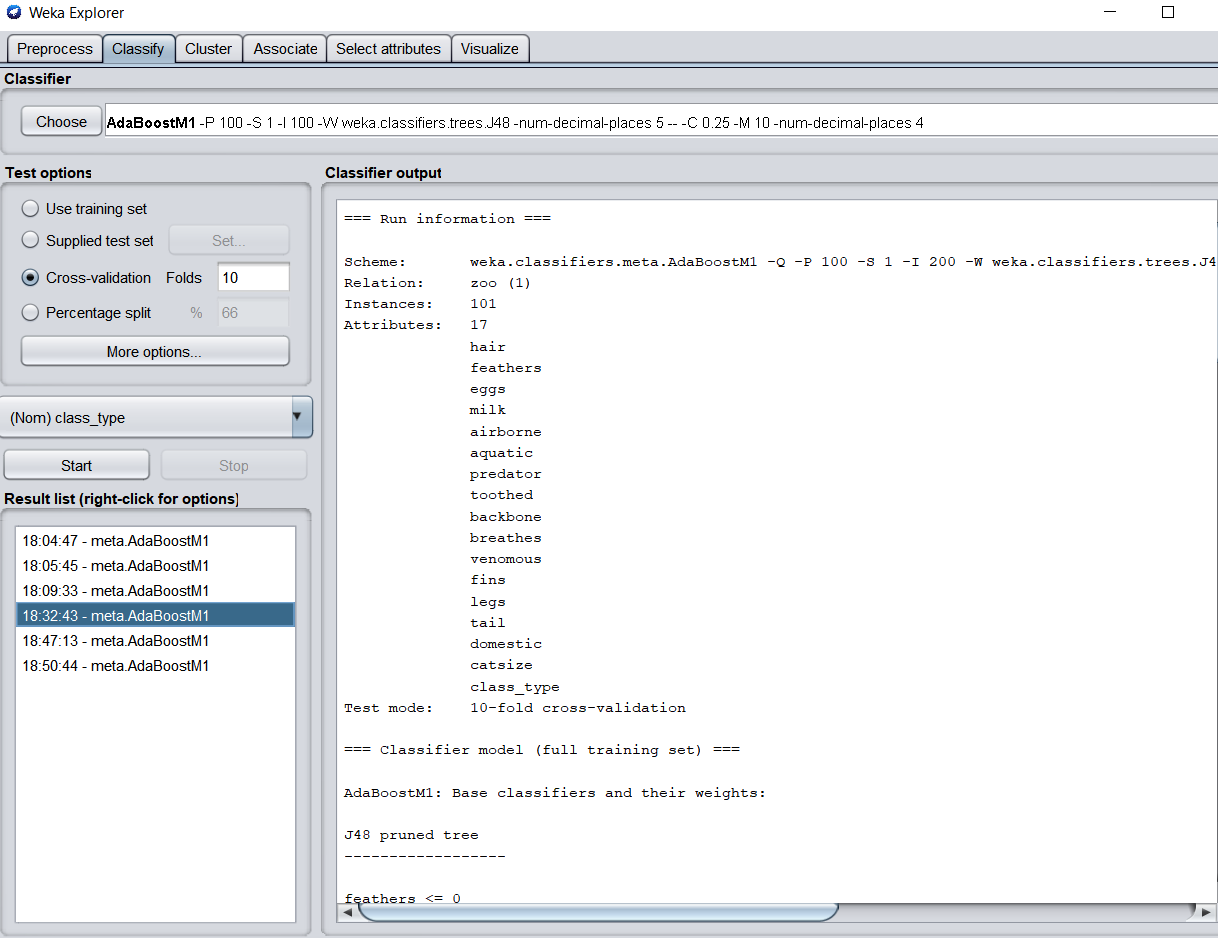
* The parameter UseResampling whether set to True of False is not affecting the result much, provided all other parameters are same.
* If we put all the parameters constant and just increase the number of iterations, the accuracy is increased little (just 2%)
* But after increasing from 100 to 200, iterations, nothing has changed. This is shown with rows in Blue.
* So after a certain threshold, the accuracy and RMSE isn’t affected by number of iterations.
* Optimal Setting : (Best Model is in Green) As we took a decent number of Folds and Minimum Number of Objects and set the Error Pruning, we got higher accuracy. No Subtree raising is performed and thus it yielded good results.
* one of the folds is use for pruning set out of the 3 in optimal case.

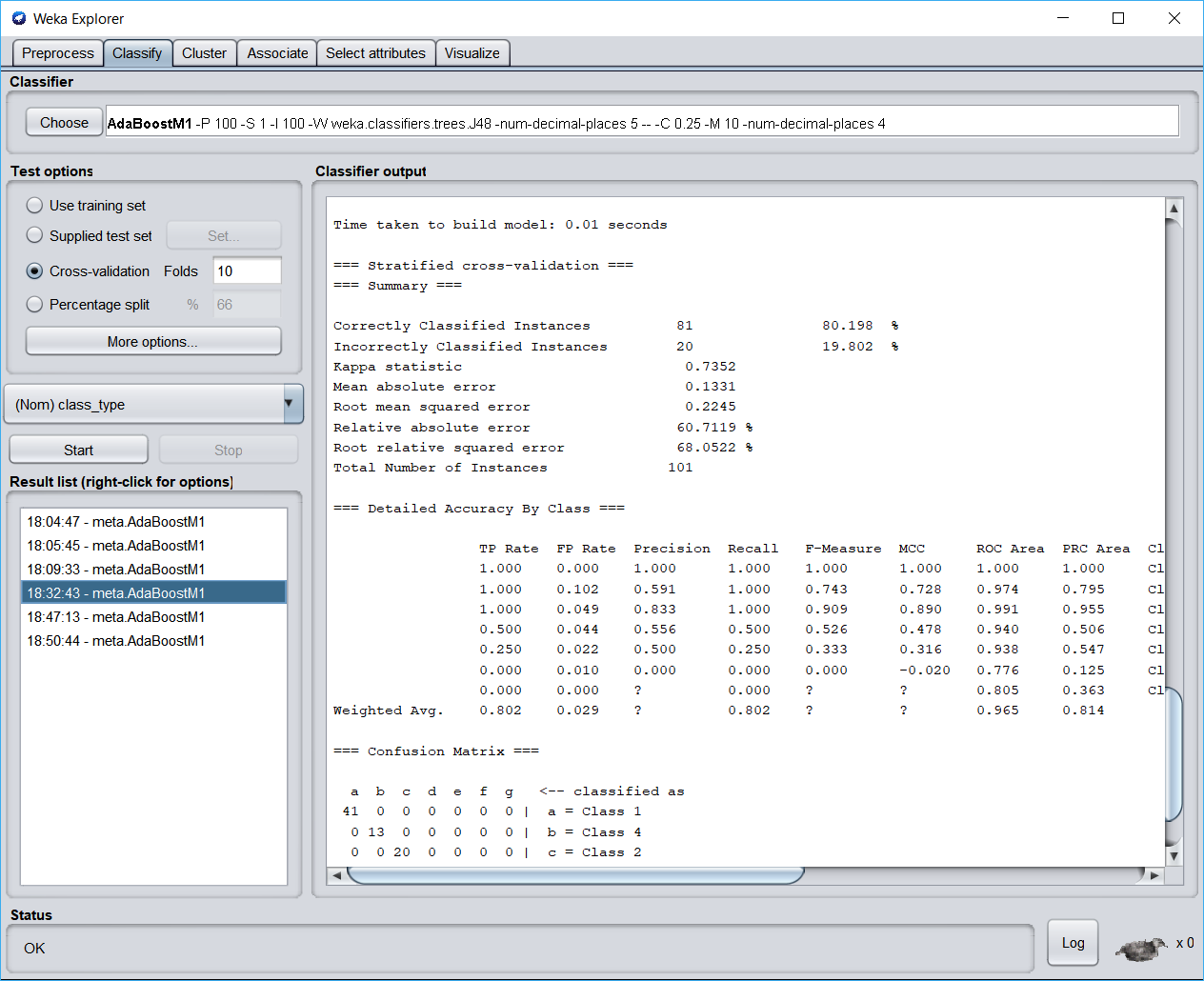
The model with best accuracy is :





The model with least accuracy is :





One example of a tree is :

J48 pruned tree

------------------

feathers <= 0

| milk <= 0

| | toothed <= 0

| | | hair <= 0: Class 7 (10.0/2.0)

| | | hair > 0: Class 6 (5.0)

| | toothed > 0

| | | aquatic <= 0: Class 3 (3.0)

| | | aquatic > 0

| | | | breathes <= 0: Class 4 (6.0)

| | | | breathes > 0: Class 5 (2.0)

| milk > 0: Class 1 (23.0)

feathers > 0: Class 2 (19.0)

Number of Leaves : 7

Size of the tree : 13

Weight: 2.76

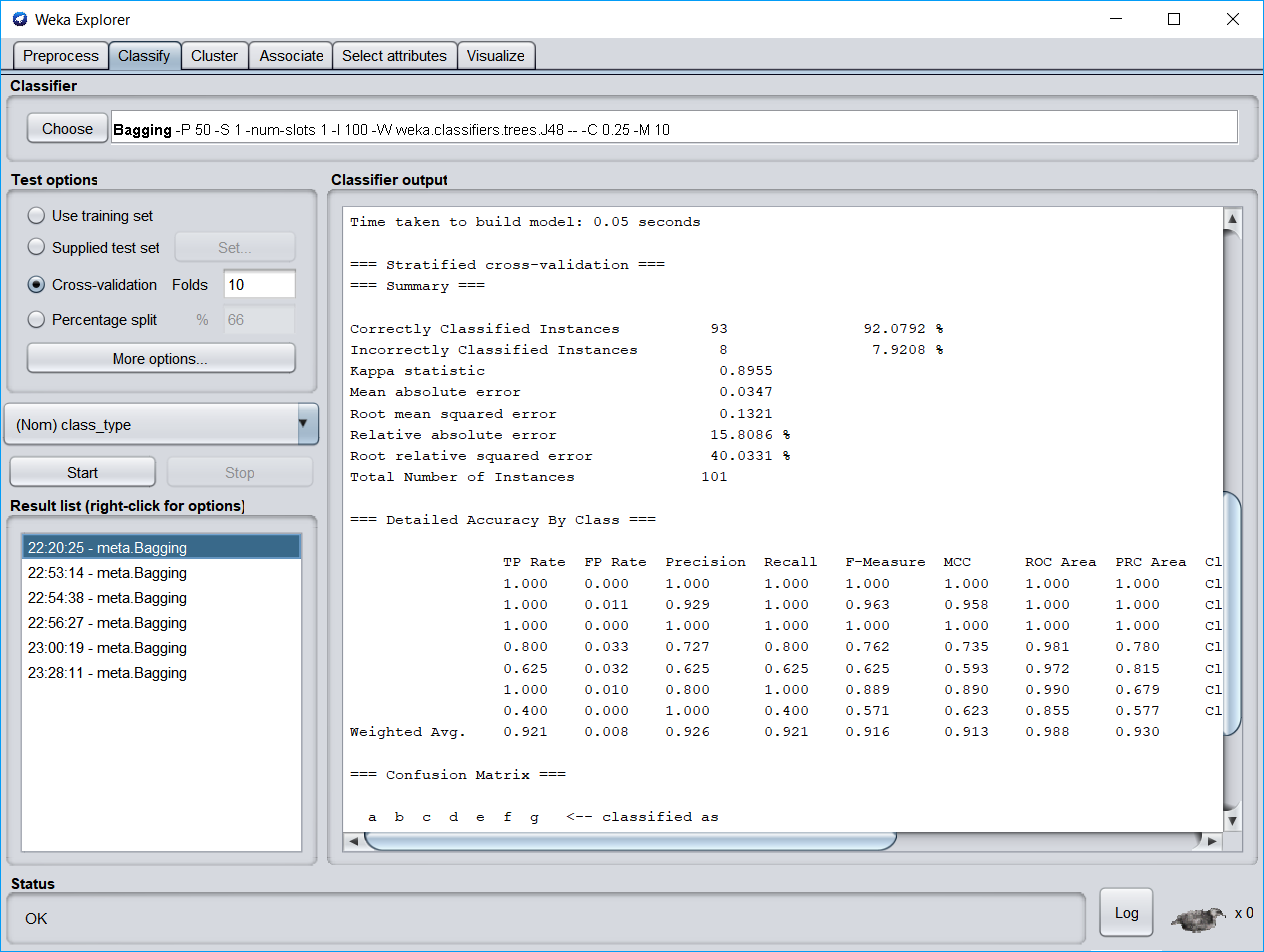
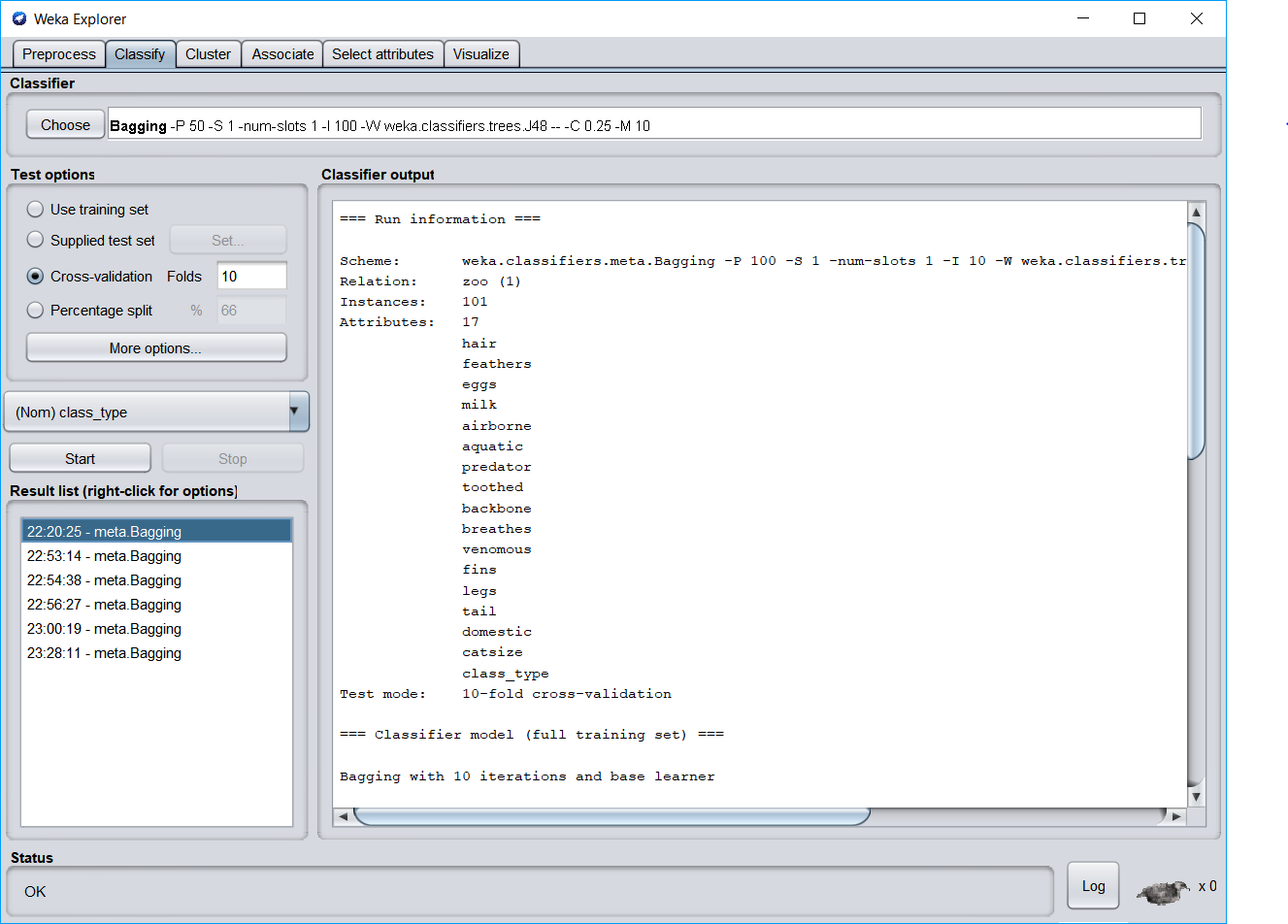
2. J48 on Bagging

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Bagging Specific parameters | | | J48 specific parameters | | | Evaluation Metrics | |
| Capabilities | NumIterations | BagSizePercent | ReducedErrorPruning | NumFolds | MinNumObj | Accuracy | RMSE |
| true | 10 | 100 | True | 3 | 2 | 92.07 | 0.13 |
| True | 30 | 80 | True | 5 | 5 | 88.11 | 0.15 |
| False | 5 | 50 | False | 3 | 10 | 77.27 | 0.21 |
| True | 200 | 80 | true | 6 | 15 | 74.25 | 0.23 |
| False | 100 | 50 | False | 3 | 10 | 83.16 | 0.2 |
| False | 200 | 50 | False | 3 | 10 | 83.16 | 0.2 |
|  |  |  |  |  |  |  |  |

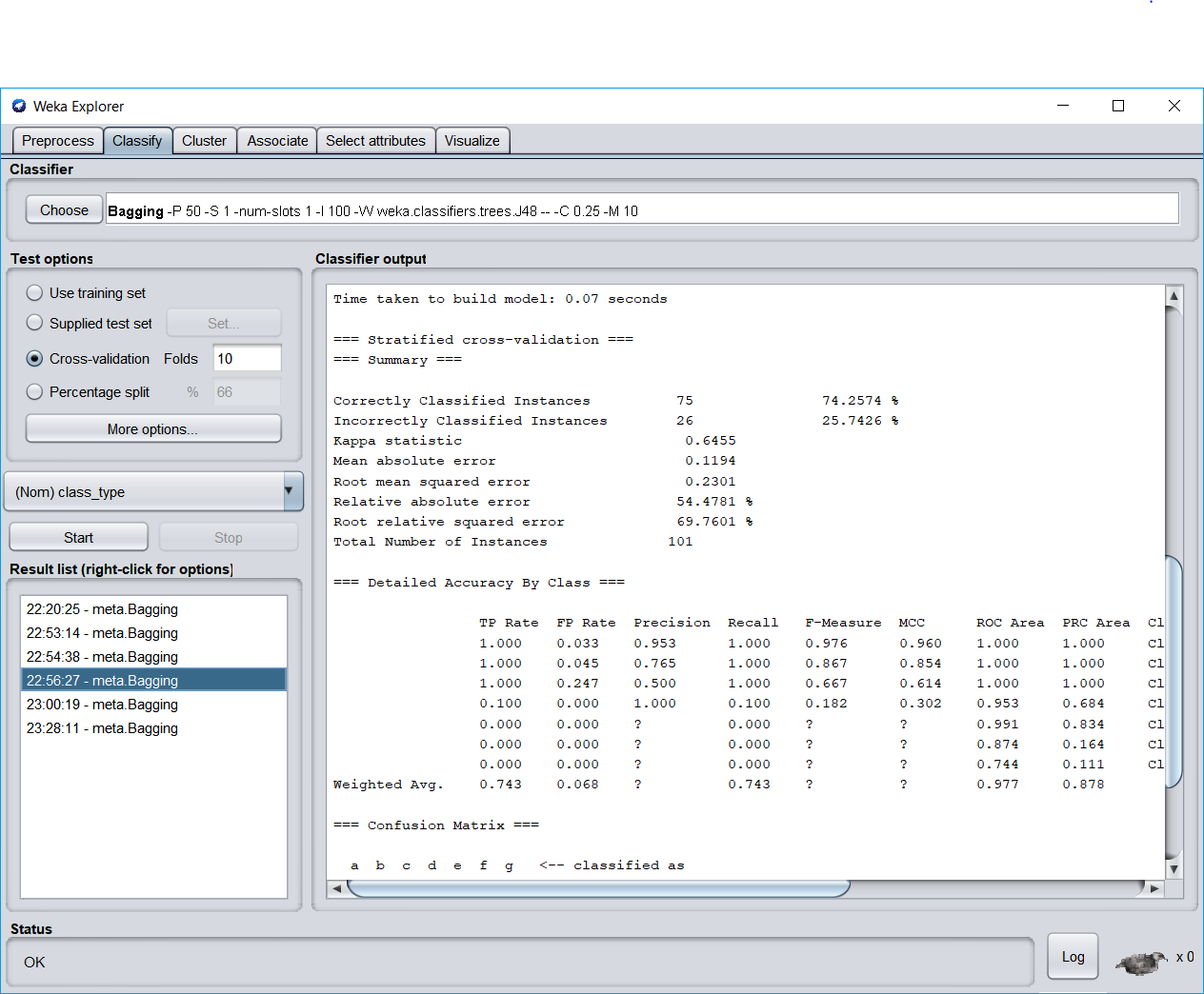
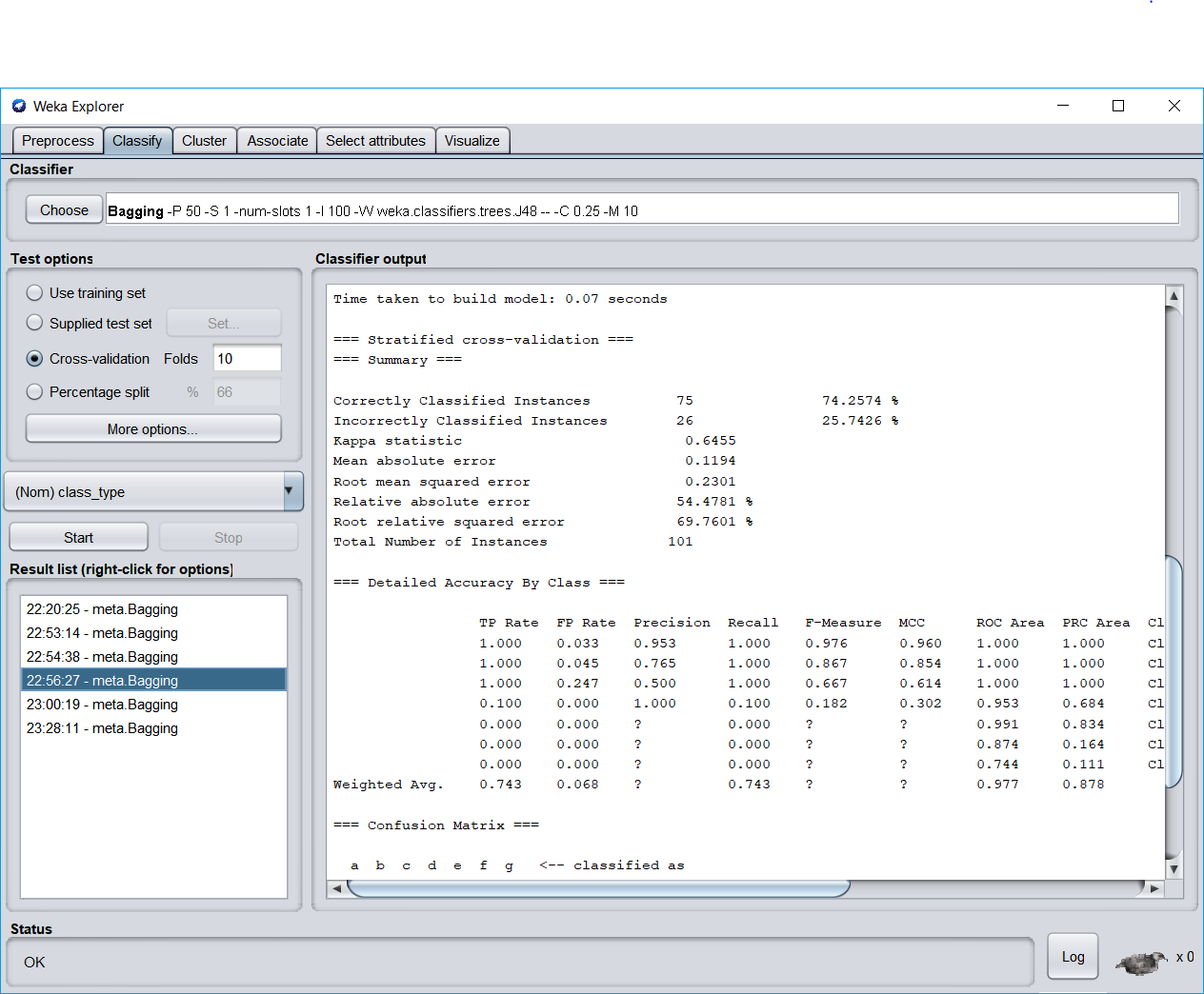
Observations :

* The model with best accuracy is in green and the one with least is in red.
* As we keep increasing the iterations, after a point, no further improvement in accuracy is seen.
* Even the F1 measures didn’t improve after certain number of iterations.
* Optimal setting : Bag Percent is one of the important reason for the accuracy being high, Since it’s set to maximum -100, we got good results.

The model with best accuracy :

****

The model which has least accuracy is :

****

AdaBoost V/s Bagging (Using J48) :

* J48 creates Binary trees and is one of the popular and powerful decision tree method.
* Using J48 on both AdaBoost and Bagging, the following results were yielded :
* The performance of Adaboost is better compared to Boosting.
* We can see that from best and least accuracies >
* Accuracy-best-Adaboost – 95.04 (RMSE – 0.11)
* Accuracy – best – Bagging – 92.07 (RMSE -0.13)
* The reason for AdaBoost being better than Boosting is :
* In AdaBoost, after a level of training and testing, the misclassified points are sampled more than the correctly sampled ones and the algorithm ir re-run. This processes can be done many times till a limit is reached. (Or till certain accuracy is achieved.)
* Where as in bagging, samples are taken out and separate classifiers are applied on thse set of samples and later all of them are merged. Here, more priority is not given to the ones misclassified.
* Hence, the algorithm wouldn’t care to see the mistakes and correct them in further iterations like AdaBoost.
* Thus, J48 works well with AdaBoost than Bagging.
* The reason for choosing J48 as first method to experiment is that it’s a very good decision tree method with it’s standards.
* Considering the size of the dataset, in AdaBoost, resampling is done only for few instance, so there are less chances for the model to overfit, the size being small.
* In Bagging on this small dataset, many samples might be generated and overfitting might happen too.

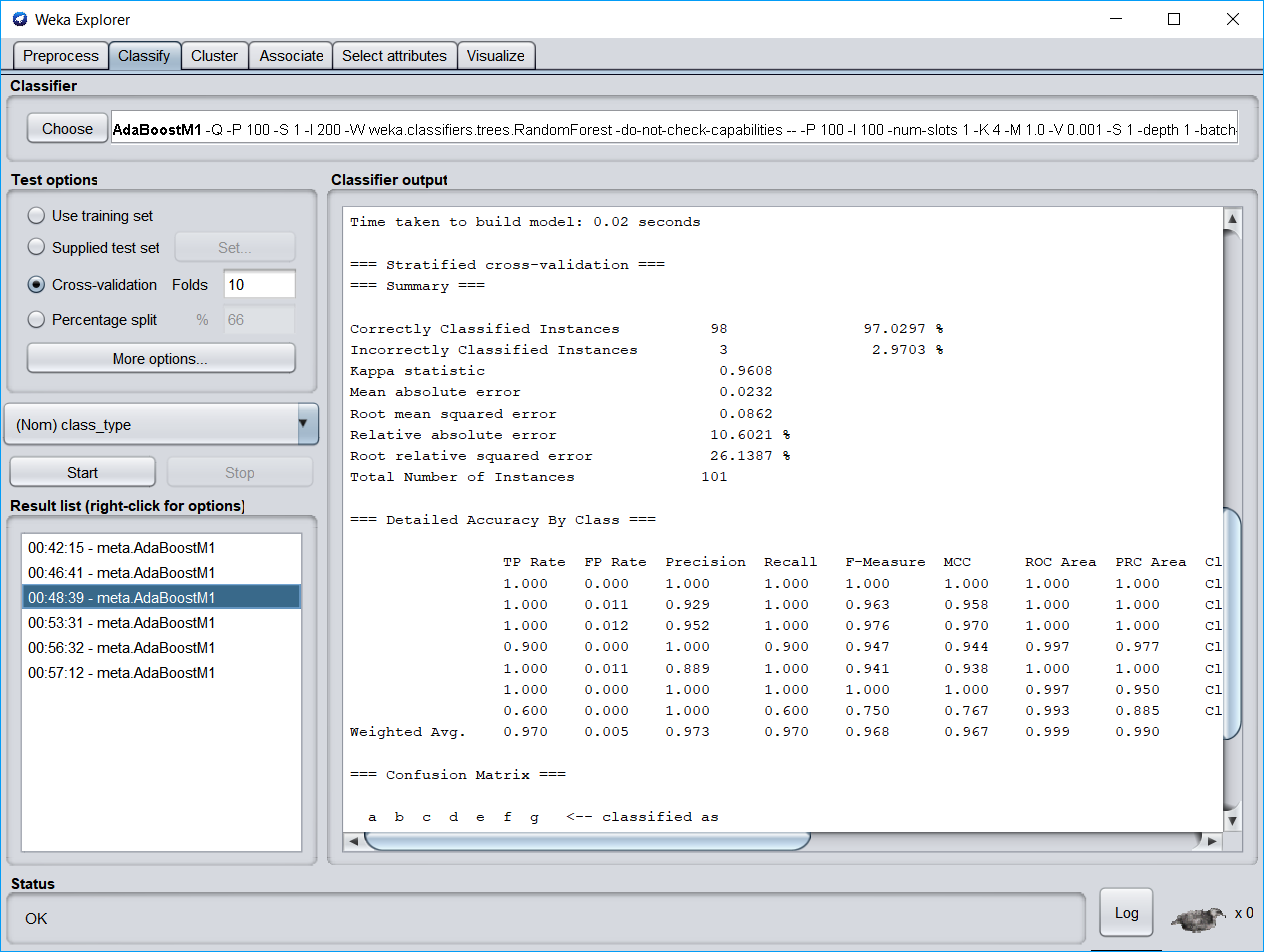
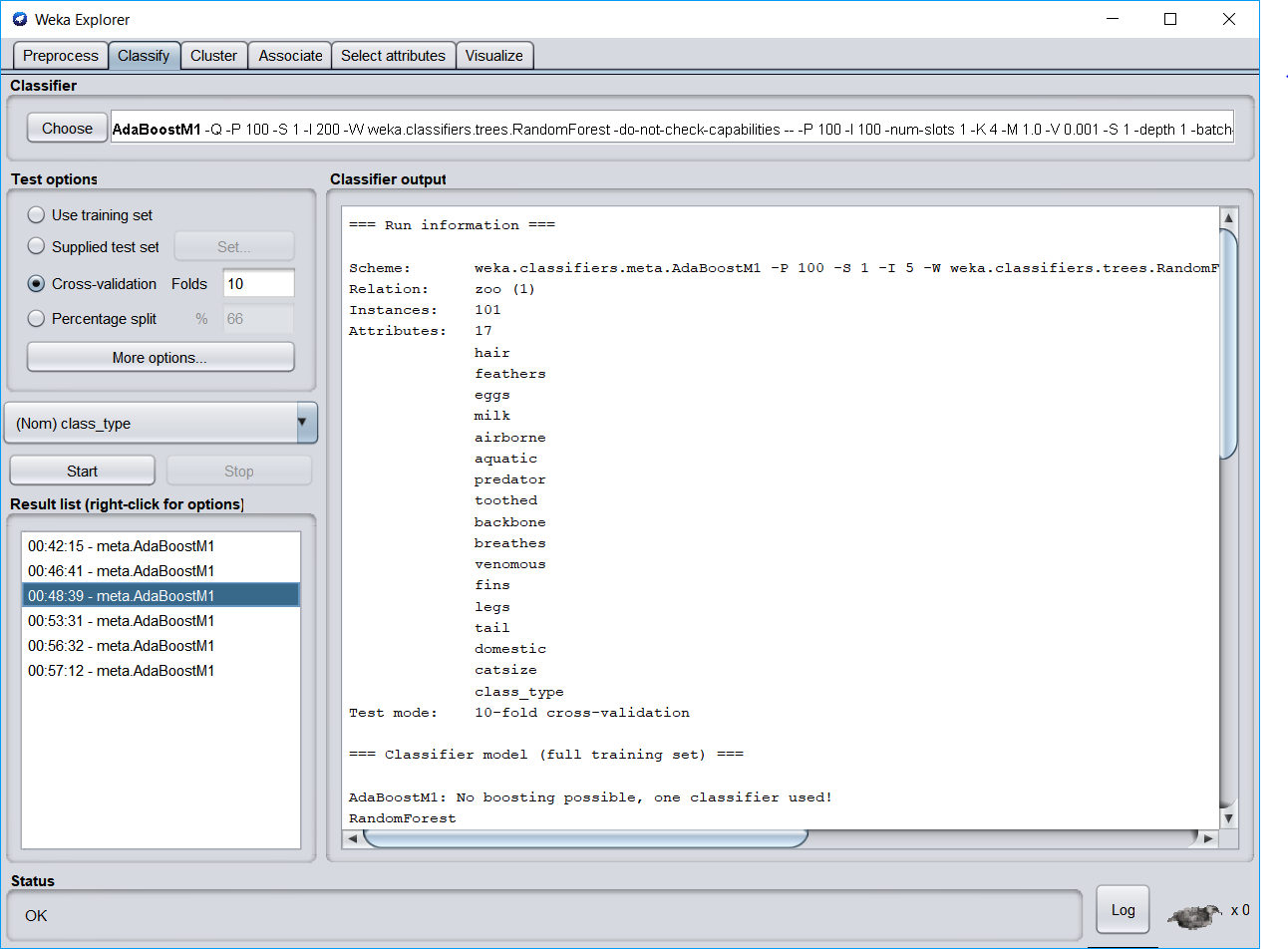
3. Random Forest on AdaBoost

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Adaboost Specific parameters | | | Random specific parameters | | | Evaluation Metrics | |
| Capabilities | NumIterations | UseResampling | BagSizePercent | MaxDepth | NumFeatures | Accuracy | RMSE |
| true | 10 | true | 100 | 4 | 2 | 95.04 | 0.11 |
| True | 30 | true | 50 | 7 | 4 | 96.03 | 0.1 |
| False | 5 | false | 50 | 7 | 4 | 97.02 | 0.08 |
| True | 200 | true | 50 | 1 | 4 | 79.20 | 0.22 |
| False | 100 | false | 50 | 7 | 4 | 97.02 | 0.08 |
| False | 200 | false | 50 | 7 | 4 | 97.02 | 0.08 |
|  |  |  |  |  |  |  |  |

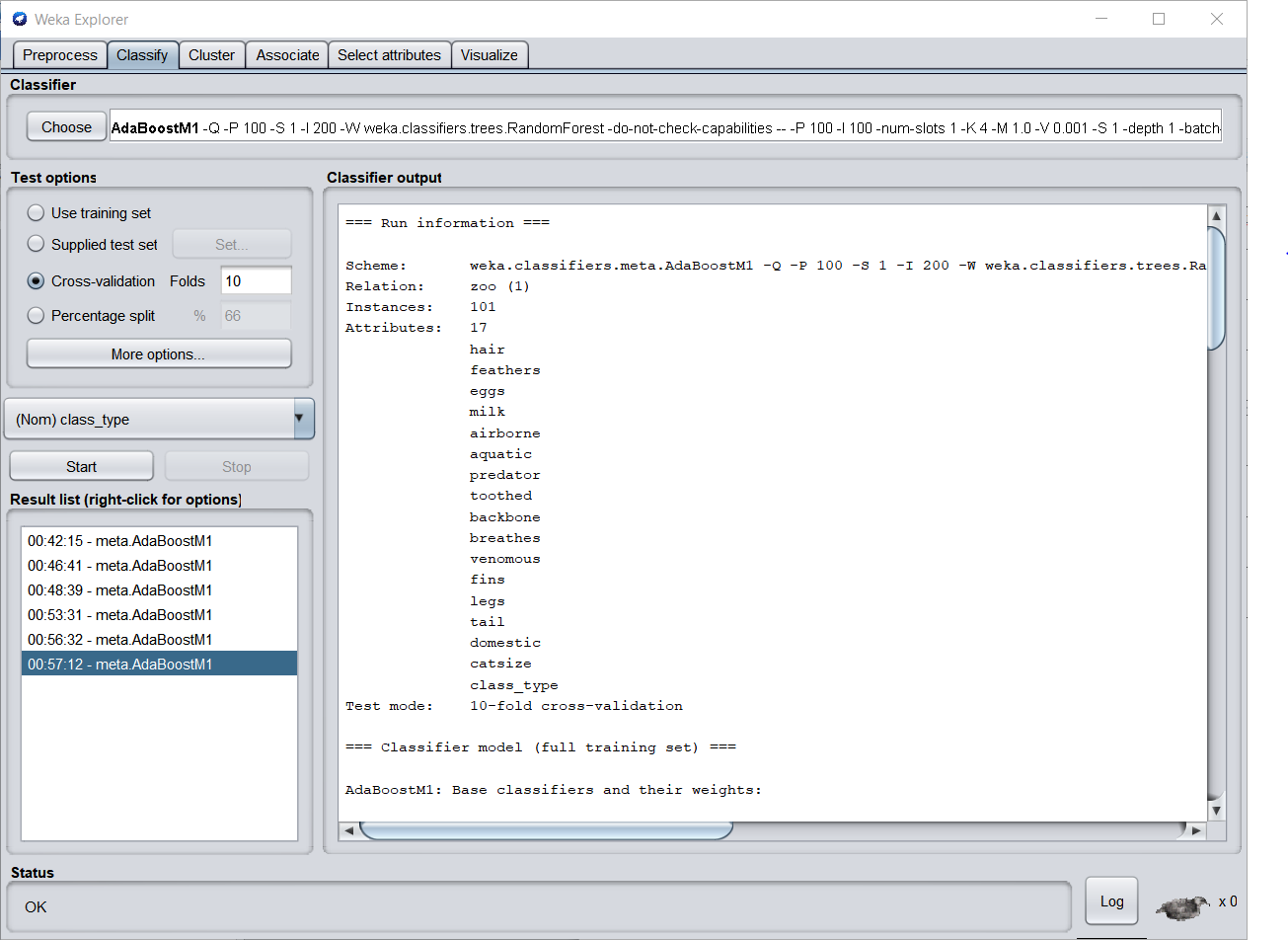
Observations :

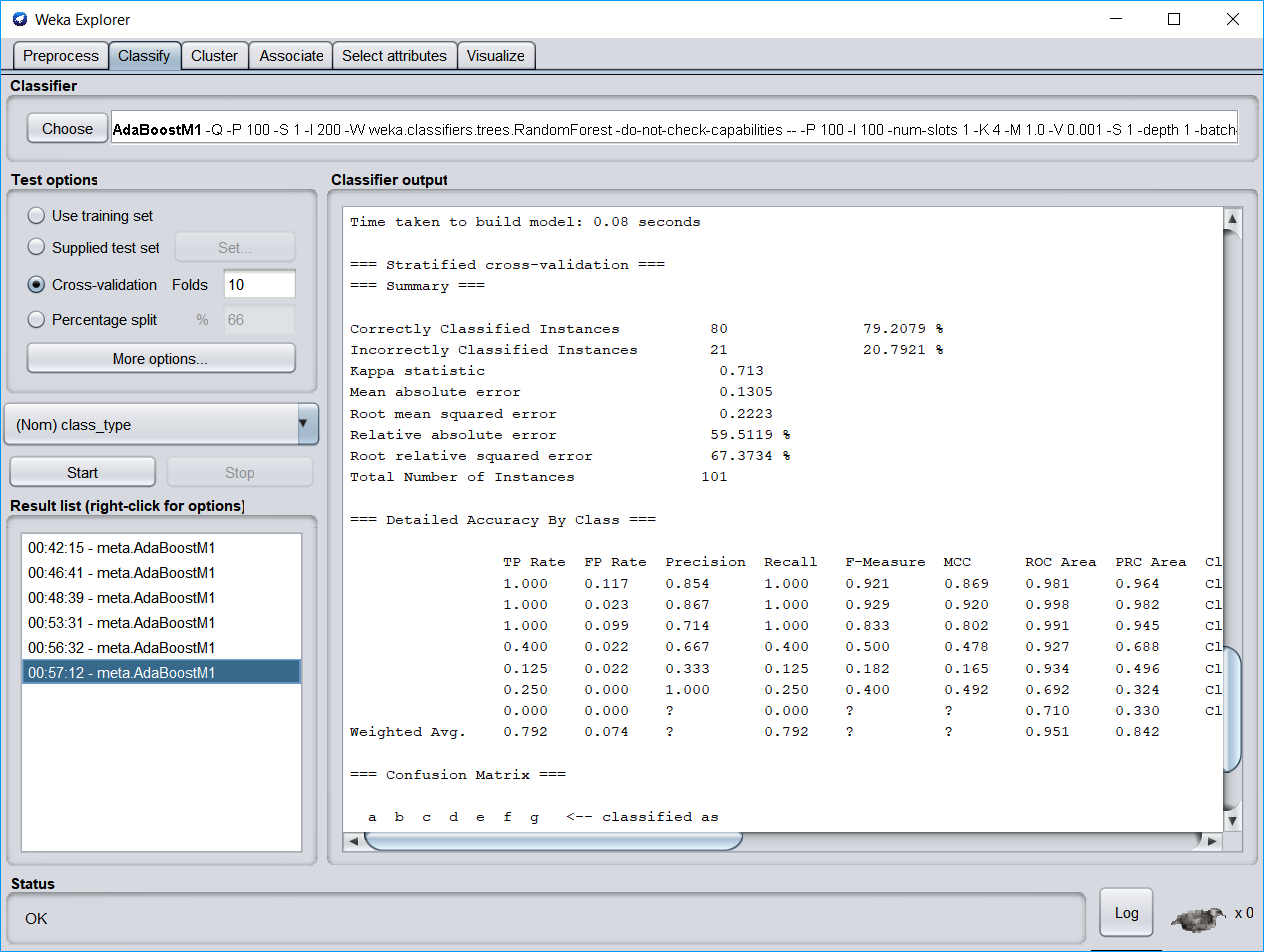
* The Bagpercent is one of the important parameter in Bagging as it’s the size of training set.
* Keeping it to the fullest might cause overfitting.
* As we keep increasing the number of iterations and keep other parameters constant, there is no much difference in accuracy after a threshold is reached.
* After 5-10 iterations, the model doesn’t improve anymore.
* Since the dataset is small, minimum number of depth in trees is enough.
* The numFeatures is very important parameter which controls the possible splits at each point.
* Since dataset is not that huge, this parameter is not given huge values.

The model with best accuracy :



The model with least accuracy :





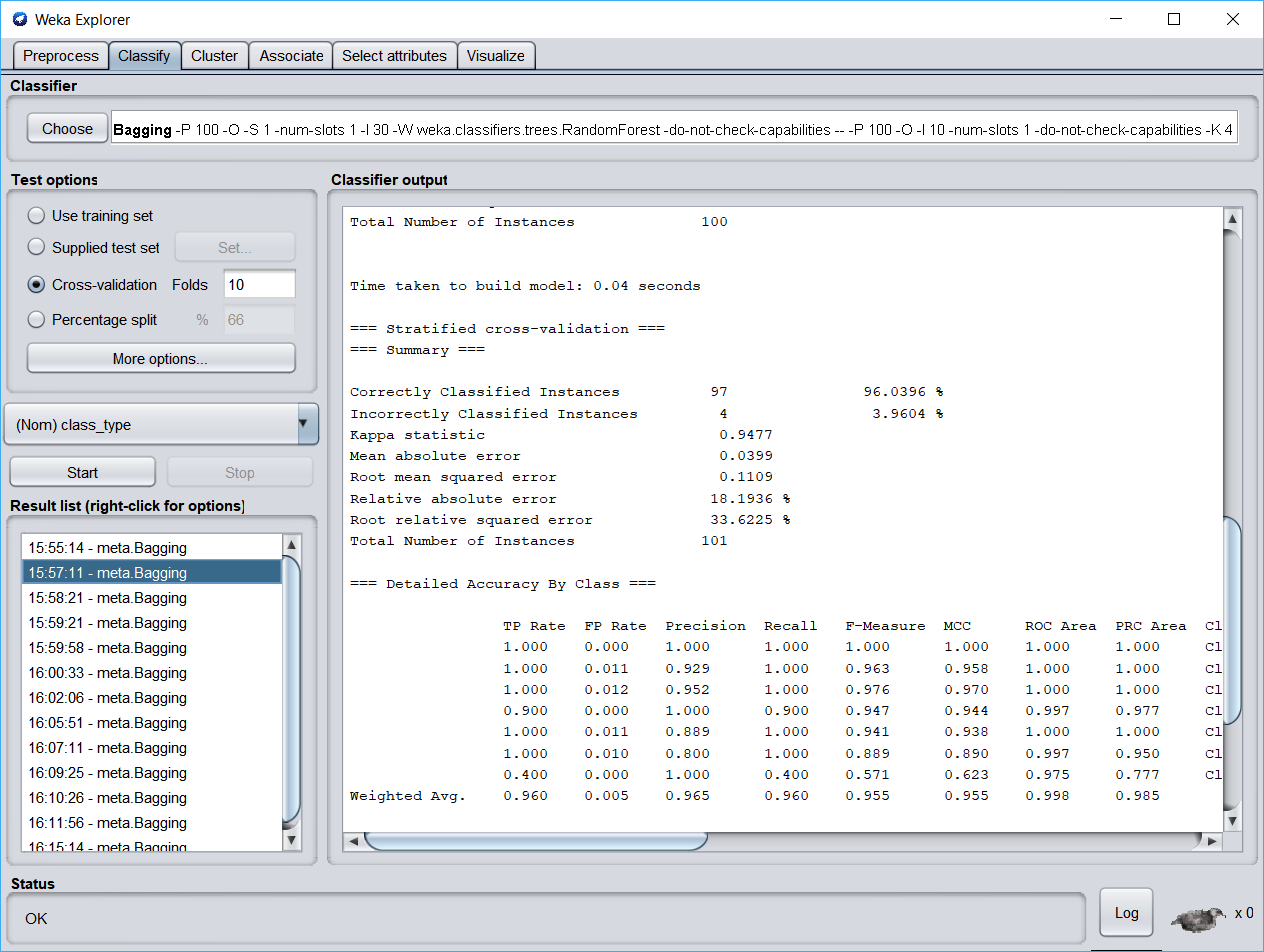
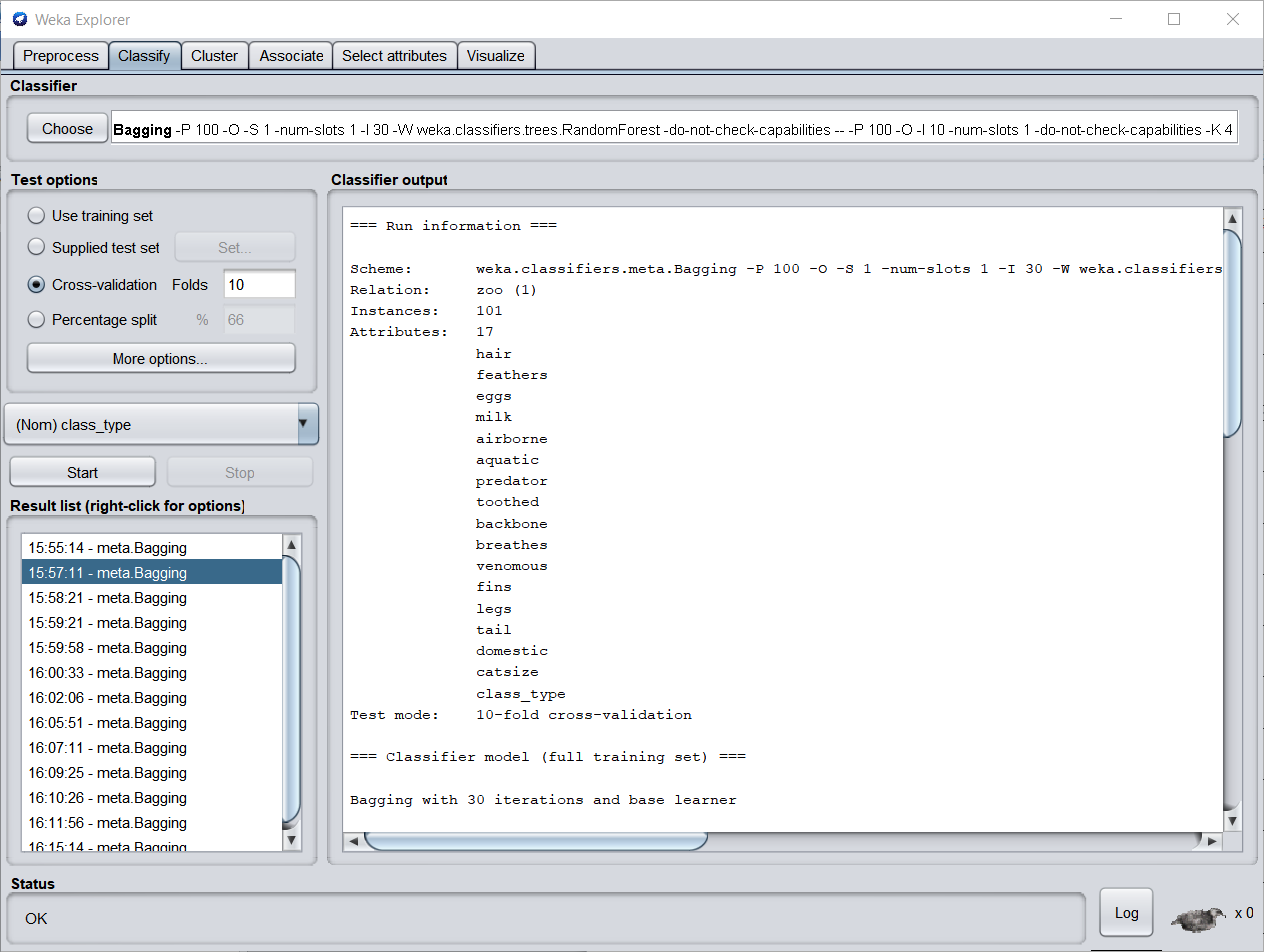
4. Random Forest on Bagging

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Bagging Specific parameters | | | Random forest specific parameters | | | Evaluation Metrics | |
| Capabilities | NumIterations | CalcOutOfbag | BagSizePercent | MaxDepth | NumFeatures |  |  |
| true | 10 | true | 100 | 4 | 2 | 94.05 | 0.13 |
| True | 30 | true | 50 | 7 | 4 | 96.03 | 0.11 |
| True | 30 | true | 10 | 7 | 4 | 88.11 | 0.18 |
| True | 30 | true | 20 | 7 | 4 | 92.07 | 0.16 |
| True | 30 | true | 40 | 7 | 4 | 92.07 | 0.16 |
| True | 30 | true | 100 | 7 | 4 | 93.06 | 0.14 |
| False | 5 | false | 50 | 7 | 4 | 96.03 | 0.11 |
| False | 100 | false | 50 | 7 | 4 | 96.03 | 0.11 |
| False | 200 | false | 50 | 7 | 4 | 96.03 | 0.11 |
| False | 2 | false | 10 | 7 | 4 | 85.14 | 0.18 |
| false | 5 | false | 10 | 3 | 4 | 89.10 | 0.18 |
| True | 200 | true | 50 | 1 | 4 | 96.03 | 0.11 |
| false | 5 | false | 10 | 1 | 4 | 71.28 | 0.25 |

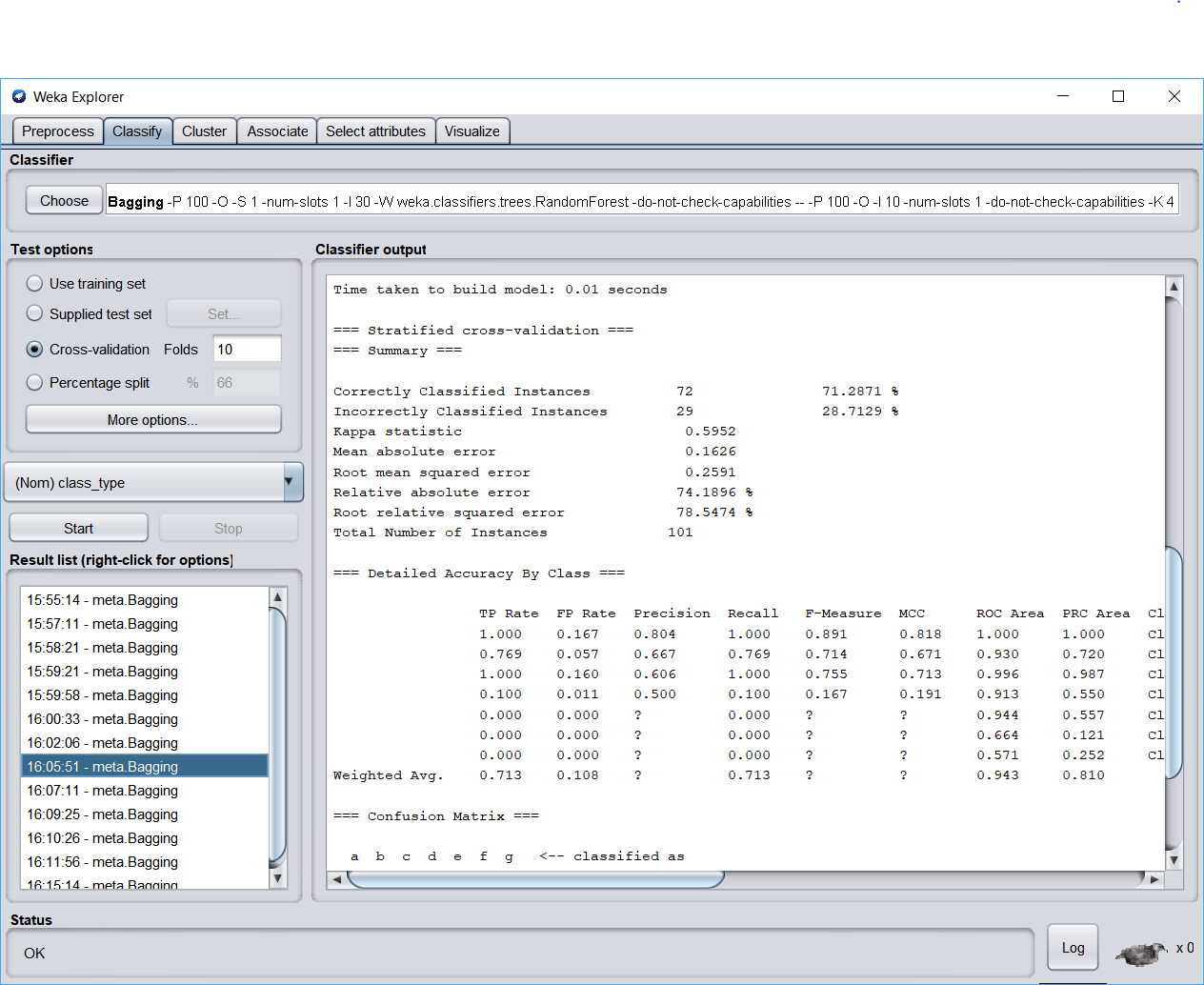
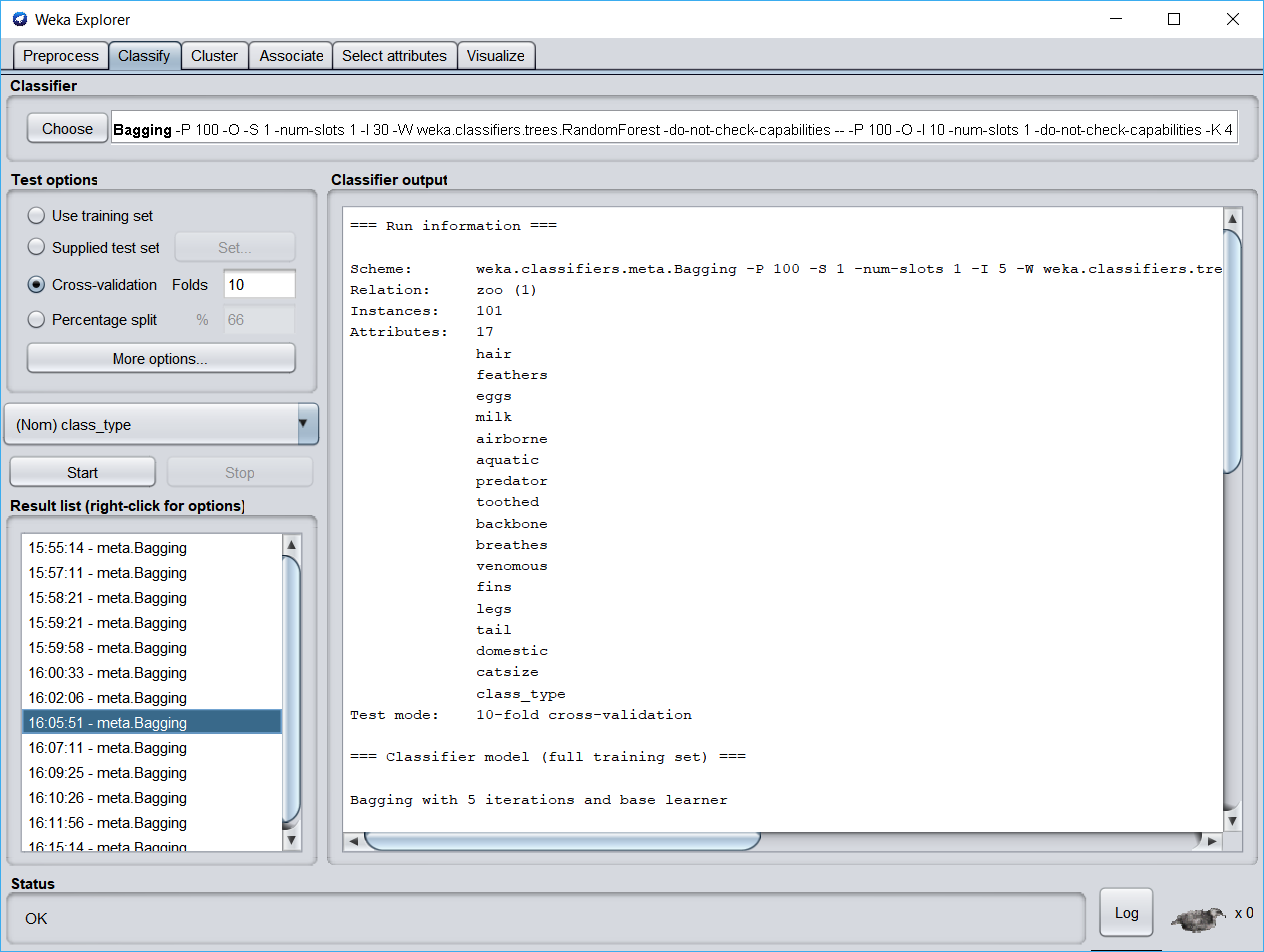
Observations :

* To get an understanding in this experiment, many models had to be done, because by changing few parameters, didn’t yield good change that can be notes of.
* Many experiments are done by changing the number of iterations and keeping other parameters constant.
* Similar to J48, increasing the number of iterations after 5-10, didn’t afftect the model much.
* The optimal setting for bagSizePercent is 50 to 100.
* Less than 50% would be too less for a model that too dataset size being less.
* The numFeatures is very important parameter which controls the possible splits at each point.
* Since dataset is not that huge, this parameter is not given huge values

The model with best accuracy :



The model with least accuracy :



AdaBoost V/s Bagging (on Random Forest):

* Random Forest is an improvement upon bagged decision trees that disrupts the greedy splitting algorithm during tree creation so that split points can only be selected from a random subset of the input attributes.
* Even using Random Forest, AdaBoost gave better models:
* Accuracy- best – Adaboost – 97.02 (RMSE – 0.08)
* Accuracy – best – Bagging – 96.03 (RMSE -0.11)
* The F-measure values are also aligned with the variation in accuracy.
* Similar reasons given for J48 make sense here as well as we just changed the learner.

1. **Consider a grid environment with four rows and five columns. Denote by (i,j) the grid position in the ith column from the left, and jth row from the bottom. Suppose the agent can move in this environment horizontally or vertically, but not on the diagonal. Suppose the Start state is in the lower left corner (1,1), and suppose the Goal is in the upper right corner (5,4).**
   * **If the reward R on reaching the goal is 100, all other rewards are 0, and γ = 0.9, manually or via a simulation calculate**
     + **(10 pts.) Q\*(x,a) for the start state and for state (3,2) only,**

Solution :

The table below is the initial Q-table.

Initially all values are 0.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| State/ Action | Up | Down | Left | Right |
| 1,1 | 0 | 0 | 0 | 0 |
| 1,2 | 0 | 0 | 0 | 0 |
| 1,3 | 0 | 0 | 0 | 0 |
| 1,4 | 0 | 0 | 0 | 0 |
| 2,1 | 0 | 0 | 0 | 0 |
| 2,2 | 0 | 0 | 0 | 0 |
| 2,3 | 0 | 0 | 0 | 0 |
| 2,4 | 0 | 0 | 0 | 0 |
| 3,1 | 0 | 0 | 0 | 0 |
| 3,1 | 0 | 0 | 0 | 0 |
| 3,3 | 0 | 0 | 0 | 0 |
| 3,4 | 0 | 0 | 0 | 0 |
| 4,1 | 0 | 0 | 0 | 0 |
| 4,2 | 0 | 0 | 0 | 0 |
| 4,3 | 0 | 0 | 0 | 0 |
| 4,4 | 0 | 0 | 0 | 0 |
| 5,1 | 0 | 0 | 0 | 0 |
| 5,2 | 0 | 0 | 0 | 0 |
| 5,3 | 0 | 0 | 0 | 0 |
| 5,4 | 0 | 0 | 0 | 0 |

Calculations:   
NOTE : the states and actions marked with BLUE are not possible and are represented as ‘NA’ in the further updated Q-tables.

All the calculations are based on the formula :

Q(X,a) = R(X) + ꞅ \*max (a’) { Q(X’, a’)} \* P(X’ | Q(X, a))

Q((5,4), up

Q((5,4), down)= R(5,4) + ꞅ \*max { (Q((5,4), Up), (Q(5,3), down), (Q((4,4), left)), (Q((5,4), right)) } \* P((5,4)| (5,4), down)

= 100 + 0.9 \* (0) \* 0

= 100 + 0

= 100

Q((5,4), left)= R(5,4) + ꞅ \*max { (Q((5,4), Up), (Q(5,3), down), (Q((4,4), left)), (Q((5,4), right)) } \* P((5,4)| (5,4), left)

= 100 + 0.9 \* (0) \* 0

= 100 + 0

= 100

Q((5,4), right

------------------------------------------------------------------------------------------------------

Q((5,3), Up)= R(5,3) + ꞅ \*max { (Q((5,4), Up), (Q(5,2), down), (Q((4,3), left)), (Q((5,3), right)) } \* P((5,4)| (5,3), Up)

= 0 + 0.9 \* (100) \* 1

= 90

Q((5,3), down)= R(5,3) + ꞅ \*max { (Q((5,4), Up), (Q(5,2), down), (Q((4,3), left)), (Q((5,3), right)) } \* P((5,4)| (5,3), down)

= 0 + 0.9 \* (100) \* 0

= 0

Q((5,3), left)= R(5,3) + ꞅ \*max { (Q((5,4), Up), (Q(5,3), down), (Q((4,4), left)), (Q((5,4), right)) } \* P((5,4)| (5,3), left)

= 0+ 0.9 \* (100) \* 0

=0

Q((5,3), right

Q((4,4), Up)

Q((4,4), down)= R(4,4) + ꞅ \*max { (Q((4,4), Up), (Q(4,3), down), (Q((3,4), left)), (Q((5,4), right)) } \* P((5,4)| (4,4), down)

= 0 + 0.9 \* (100) \* 0

= 0

Q((4,4), left)= R(4,4) + ꞅ \*max { (Q((4,4), Up), (Q(4,3), down), (Q((3,4), left)), (Q((5,4), right)) } \* P((5,4)| (4,4), left)

= 0 + 0.9 \* (100) \* 0

= 0

Q((4,4), right)= R(4,4) + ꞅ \*max { (Q((4,4), Up), (Q(4,3), down), (Q((3,4), left)), (Q((5,4), right)) } \* P((5,4)| (4,4), right)

= 0 + 0.9 \* (100) \* 1

= 90

-----------------------------------------------------------------------------------------------------

Q((3,4), Up)

Q((3,4), down)= R(3,4) + ꞅ \*max { (Q((3,4), Up), (Q(3,3), down), (Q((2,4), left)), (Q((4,4), right)) } \* P((4,4)| (3,4), down)

= 0 + 0.9 \* (90) \* 0

= 0

Q((3,4), left)= R(3,4) + ꞅ \*max { (Q((3,4), Up), (Q(3,3), down), (Q((2,4), left)), (Q((4,4), right)) } \* P((4,4)| (3,4), left)

= 0 + 0.9 \* (90) \* 0

= 0

Q((3,4), right)= R(3,4) + ꞅ \*max { (Q((3,4), Up), (Q(3,3), down), (Q((2,4), left)), (Q((4,4), right)) } \* P((4,4)| (3,4), right)

= 0 + 0.9 \* (90) \* 1

= 81

------------------------------------------------------------------------------------------------------------------------------------------

Q((5,2), Up)= R(5,2) + ꞅ \*max { (Q((5,3), Up), (Q(5,1), down), (Q((4,2), left)), (Q((5,2), right)) } \* P((5,3)| (5,2), Up)

= 0 + 0.9 \* (90) \* 1

= 81

Q((5,2), down)= R(5,2) + ꞅ \*max { (Q((5,3), Up), (Q(5,1), down), (Q((4,2), left)), (Q((5,2), right)) } \* P((5,3)| (5,2), down)

= 0 + 0.9 \* (90) \* 0

= 0

Q((5,2), left)= R(5,2) + ꞅ \*max { (Q((5,3), Up), (Q(5,1), down), (Q((4,2), left)), (Q((5,2), right)) } \* P((5,3)| (5,2), left)

= 0 + 0.9 \* (90) \* 0

= 0

Q((5,2), right)

------------------------------------------------------------------------------------------------------------------------------------------

Q((5,1), Up)= R(5,1) + ꞅ \*max { (Q((5,2), Up), (Q(5,1), down), (Q((4,1), left)), (Q((5,1), right)) } \* P((5,2)| (5,1), up)

= 0 + 0.9 \* (81) \* 1

= 72.9

Q((5,1), down)

Q((5,1), left)= R(5,1) + ꞅ \*max { (Q((5,2), Up), (Q(5,1), down), (Q((4,1), left)), (Q((5,1), right)) } \* P((5,2)| (5,1), left)

= 0 + 0.9 \* (81) \* 0

= 0

Q((5,1), right)

Q((4,3), Up)= R(4,3) + ꞅ \*max { (Q((4,4), Up), (Q(4,2), down), (Q((3,3), left)), (Q((5,3), right)) } \* P((4,4)| (4,3), Up)

= 0 + 0.9 \* (90) \* 1

= 81

Q((4,3), down)= R(4,3) + ꞅ \*max { (Q((4,4), Up), (Q(4,2), down), (Q((3,3), left)), (Q((5,3), right)) } \* P((4,4)| (4,3), down)

= 0 + 0.9 \* (90) \* 0

= 0

Q((4,3), left)= R(4,3) + ꞅ \*max { (Q((4,4), Up), (Q(4,2), down), (Q((3,3), left)), (Q((5,3), right)) } \* P((4,4)| (4,3), left)

= 0 + 0.9 \* (90) \* 0

= 0

Q((4,3), right)= R(4,3) + ꞅ \*max { (Q((4,4), Up), (Q(4,2), down), (Q((3,3), left)), (Q((5,3), right)) } \* P((5,3)| (4,3), right)

= 0 + 0.9 \* (90) \* 1

= 81

Observing the pattern, we can see that at every layer we move away from the goal, we can see a decrease by factor 0.9.

This decrease is done in 2 aspects:

From the goal, as we move left and down, the Q values decrease since we are going away from the goal. (considering goal is on top right).

Similar computations are done for futher states and actions and thus the table is updated below

The updated Q Table is :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| State/ Action | Up | Down | Left | Right |
| 1,1 | 47.82 | NA | NA | 47.82 |
| 1,2 | 53.14 | 0 | NA | 53.14 |
| 1,3 | 59.04 | 0 | NA | 59.04 |
| 1,4 | NA | 0 | NA | 65.61 |
| 2,1 | 53.14 | NA | 0 | 53.14 |
| 2,2 | 59.04 | 0 | 0 | 59.04 |
| 2,3 | 65.61 | 0 | 0 | 65.61 |
| 2,4 | NA | 0 | 0 | 72.9 |
| 3,1 | 59.04 | NA | 0 | 59.04 |
| 3,2 | 65.61 | 0 | 0 | 65.61 |
| 3,3 | 72.90 | 0 | 0 | 72.90 |
| 3,4 | NA | 0 | 0 | 81 |
| 4,1 | 65.61 | NA | 0 | 65.61 |
| 4,2 | 72.90 | 0 | 0 | 72.90 |
| 4,3 | 81 | 0 | 0 | 81 |
| 4,4 | NA | 0 | 0 | 90 |
| 5,1 | 72.9 | NA | 0 | NA |
| 5,2 | 81 | 0 | 0 | NA |
| 5,3 | 90 | 0 | 0 | NA |
| 5,4 | NA | 100 | 100 | NA |

Therefore , for start state (1,1):

Q((1,1), UP) = 47.82

Q((1,1), down) = (Bounces back - NA)

Q((1,1), left) = (Bounces back - NA)

Q((1,1), right) = 47.82

Therefore , for start state (3,2):

Q((1,1), UP) =65.61

Q((1,1), down) = 0

Q((1,1), left) = 0

Q((1,1), right) = 65.61

NOTE : these are the values after initial iteration.

After doing another iteration because there are many unknown Q-values (zero’s), we get updated values.

The calculation are done in a similar way as shown above.

They are :

Therefore , for start state (1,1):

Q((1,1), UP) = 47.82

Q((1,1), down) = (Bounces back - NA)

Q((1,1), left) = (Bounces back - NA)

Q((1,1), right) = 47.82

Therefore , for start state (3,2):

Q((1,1), UP) =65.61

Q((1,1), down) = 53.14

Q((1,1), left) = 53.14

Q((1,1), right) = 65.61

* + - **(10 pts.) V\*(x) for every state**

The maximum value in that state with a particular action

Max(a) { Q(X, a) }

Since every row in the following table represents a state,

We select the action – the value in the that is highest

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| State/ Action | Up | Down | Left | Right |  |  |  |  |  |
| 1,1 | 47.82 | NA | NA | 47.82 |  |  |  |  |  |
| 1,2 | 53.14 | 0 | NA | 53.14 |  |  |  |  |  |
| 1,3 | 59.04 | 0 | NA | 59.04 |  |  |  |  |  |
| 1,4 | NA | 0 | NA | 65.61 |  |  |  |  |  |
| 2,1 | 53.14 | NA | 0 | 53.14 |  |  |  |  |  |
| 2,2 | 59.04 | 0 | 0 | 59.04 |  |  |  |  |  |
| 2,3 | 65.61 | 0 | 0 | 65.61 |  |  |  |  |  |
| 2,4 | NA | 0 | 0 | 72.9 |  |  |  |  |  |
| 3,1 | 59.04 | NA | 0 | 59.04 |  |  |  |  |  |
| 3,2 | 65.61 | 0 | 0 | 65.61 |  |  |  |  |  |
| 3,3 | 72.90 | 0 | 0 | 72.90 |  |  |  |  |  |
| 3,4 | NA | 0 | 0 | 81 |  |  |  |  |  |
| 4,1 | 65.61 | NA | 0 | 65.61 |  |  |  |  |  |
| 4,2 | 72.90 | 0 | 0 | 72.90 |  |  |  |  |  |
| 4,3 | 81 | 0 | 0 | 81 |  |  |  |  |  |
| 4,4 | NA | 0 | 0 | 90 |  |  |  |  |  |
| 5,1 | 72.9 | NA | 0 | NA |  |  |  |  |  |
| 5,2 | 81 | 0 | 0 | NA |  |  |  |  |  |
| 5,3 | 90 | 0 | 0 | NA |  |  |  |  |  |
| 5,4 | NA | 100 | 100 | NA |  |  |  |  |  |

* + - **(10 pts.) the optimal policy.**

Optimal policy - for every state, there is no other action that gets a higher sum of discounted future rewards.

Here, those actions are indicated by using the word ‘yes’.

The other actions which are not marked ‘yes’ are not optimal to choose in that particular state.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| State/ Action | Up | Down | Left | Right |
| 1,1 | Yes |  |  | Yes |
| 1,2 | yes |  |  | yes |
| 1,3 | yes |  |  | yes |
| 1,4 |  |  |  | yes |
| 2,1 | yes |  |  | Yes |
| 2,2 | Yes |  |  | yes |
| 2,3 | yes |  |  | yes |
| 2,4 |  |  |  | yes |
| 3,1 | Yes |  |  | Yes |
| 3,1 | Yes |  |  | Yes |
| 3,3 | Yes |  |  | Yes |
| 3,4 |  |  |  | yes |
| 4,1 | yes |  |  | Yes |
| 4,2 | yes |  |  | Yes |
| 4,3 | Yes |  |  | Yes |
| 4,4 |  |  |  | yes |
| 5,1 | Yes |  |  |  |
| 5,2 | Yes |  |  |  |
| 5,3 | Yes |  |  |  |
| 5,4 |  | yes | yes |  |

* + **(10 pts. BONUS) (Implementation only) In the original environment, use Q learning to learn the optimal policy.**
  + **(10 pts. BONUS) How does the optimal policy change if another goal state is added, in the lower right corner (5,1)?**

**When we add a new goal, here is the optimal policy :**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| State/ Action | Up | Down | Left | Right |
| 1,1 |  |  |  | yes |
| 1,2 |  | yes |  | yes |
| 1,3 | yes |  |  | Yes |
| 1,4 |  |  |  | yes |
| 2,1 |  |  |  | yes |
| 2,2 |  | yes |  | Yes |
| 2,3 | yes |  |  | yes |
| 2,4 |  |  |  | yes |
| 3,1 |  |  |  | yes |
| 3,2 |  | yes |  | Yes |
| 3,3 | yes |  |  | yes |
| 3,4 |  |  |  | yes |
| 4,1 |  |  |  | yes |
| 4,2 |  | yes |  | yes |
| 4,3 | Yes |  |  | Yes |
| 4,4 |  |  |  | Yes |
| 5,1 |  |  |  |  |
| 5,2 |  | Yes |  |  |
| 5,3 | yes |  |  |  |
| 5,4 |  |  |  |  |

In simple words, it can be explained like this :

1,1 🡪 2,1 🡪 3,1 🡪 4,1 🡪 5,1

Where starting from goal, 4 steps right reaches the goal.

* + **(10 pts. BONUS) What happens if a very low-reward state (R = -100) is instead defined in (5,1) instead of this second goal?**

The Q- table would be

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| State/ Action | Up | Down | Left | Right |
| 1,1 | 47.82 | NA | NA | 47.82 |
| 1,2 | 53.14 | 43.05 | NA | 53.14 |
| 1,3 | 59.04 | 47.83 | NA | 59.04 |
| 1,4 | NA | 53.14 | NA | 65.61 |
| 2,1 | 53.14 | NA | 43.05 | 53.14 |
| 2,2 | 59.04 | 47.83 | 47.83 | 59.04 |
| 2,3 | 65.61 | 53.14 | 53.14 | 65.61 |
| 2,4 | NA | 59.05 | 59.05 | 72.9 |
| 3,1 | 59.04 | NA | 47.83 | 59.04 |
| 3,2 | 65.61 | 53.14 | 53.14 | 65.61 |
| 3,3 | 72.90 | 59.05 | 59.05 | 72.90 |
| 3,4 | NA | 65.61 | 65.61 | 81 |
| 4,1 | 65.61 | NA | 53.14 | -24.39 |
| 4,2 | 72.90 | 59.04 | 59.04 | 72.90 |
| 4,3 | 81 | 65.61 | 65.61 | 81 |
| 4,4 | NA | 72.9 | 72.9 | 90 |
| 5,1 | -27.1 | NA | -40.96 | NA |
| 5,2 | 81 | -24.39 | 65.61 | NA |
| 5,3 | 90 | 72.91 | 72.91 | NA |
| 5,4 | NA | 100 | 100 | NA |

The V(X) for every state X is the value which is highest in that row highlighted with green

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| State/ Action | Up | Down | Left | Right |
| 1,1 | 47.82 | NA | NA | 47.82 |
| 1,2 | 53.14 | 43.05 | NA | 53.14 |
| 1,3 | 59.04 | 47.83 | NA | 59.04 |
| 1,4 | NA | 53.14 | NA | 65.61 |
| 2,1 | 53.14 | NA | 43.05 | 53.14 |
| 2,2 | 59.04 | 47.83 | 47.83 | 59.04 |
| 2,3 | 65.61 | 53.14 | 53.14 | 65.61 |
| 2,4 | NA | 59.05 | 59.05 | 72.9 |
| 3,1 | 59.04 | NA | 47.83 | 59.04 |
| 3,2 | 65.61 | 53.14 | 53.14 | 65.61 |
| 3,3 | 72.90 | 59.05 | 59.05 | 72.90 |
| 3,4 | NA | 65.61 | 65.61 | 81 |
| 4,1 | 65.61 | NA | 53.14 | -24.39 |
| 4,2 | 72.90 | 59.04 | 59.04 | 72.90 |
| 4,3 | 81 | 65.61 | 65.61 | 81 |
| 4,4 | NA | 72.9 | 72.9 | 90 |
| 5,1 | -27.1 | NA | -40.96 | NA |
| 5,2 | 81 | -24.39 | 65.61 | NA |
| 5,3 | 90 | 72.91 | 72.91 | NA |
| 5,4 | NA | 100 | 100 | NA |

The optimal policy would be :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| State/ Action | Up | Down | Left | Right |
| 1,1 | yes |  |  | yes |
| 1,2 | yes |  |  | yes |
| 1,3 | yes |  |  | Yes |
| 1,4 |  |  |  | yes |
| 2,1 | yes |  |  | yes |
| 2,2 | Yes |  |  | Yes |
| 2,3 | yes |  |  | yes |
| 2,4 |  |  |  | yes |
| 3,1 | Yes |  |  | yes |
| 3,2 | yes |  |  | Yes |
| 3,3 | yes |  |  | yes |
| 3,4 |  |  |  | yes |
| 4,1 | Yes |  |  |  |
| 4,2 | yes |  |  | yes |
| 4,3 | Yes |  |  | Yes |
| 4,4 |  |  |  | Yes |
| 5,1 | Yes |  |  |  |
| 5,2 | yes |  |  |  |
| 5,3 | yes |  |  |  |
| 5,4 |  |  |  |  |

In short words, it can be represented as :

Right, right, right, up, right, up, up

This is the order from start state.