From 661e2009df11f04db31d33034cd12ce71bc0a705 Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Fri, 2 Jun 2017 21:35:42 +1000 Subject: [PATCH 01/27] Some comments about what the bug might be

src/moa/classifiers/trees/HATADWIN.java | 31 ++++++++++++++++++++++++++++++++ 1 file changed, 31 insertions(+)

diff –git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java index 844e2e7..9afcff7 100644 — a/src/moa/classifiers/trees/HATADWIN.java +++ b/src/moa/classifiers/trees/HATADWIN.java @@ -83,6 +83,11 @@ public class HATADWIN extends HoeffdingTree {

```
public void filterInstanceToLeaves(Instance inst, SplitNode myparent, int parentBranch,
        boolean updateSplitterCounts);
```

-
- ```
  public boolean isAlternate();
  ```
-
- ```
  public void setAlternate(boolean isAlternate);
  ```
- }

public static class AdaSplitNode extends SplitNode implements NewNode { @@ -100,6 +105,20 @@ public class HATADWIN extends HoeffdingTree {

```
protected Random classifierRandom;
```

- ```
  private boolean isAlternate;
  ```
-
- ```
  @Override
  ```
- ```
  public boolean isAlternate() {
  ```
- ```
      return this.isAlternate;
  ```
- ```
  }
  ```
-
- ```
  @Override
  ```
- ```
  public void setAlternate(boolean isAlternate) {
  ```
- ```
      this.isAlternate = isAlternate;
  ```
- ```
  }
  ```
-
-
  ```

- //public boolean getErrorChange() {
  //   return ErrorChange;
  //}

  @@ -320,6 +339,18 @@ public class HATADWIN extends HoeffdingTree {

  protected Random classifierRandom;

- private boolean isAlternate;
-
- @Override
- public boolean isAlternate() {
-     return this.isAlternate;
- }
-
- @Override
- public void setAlternate(boolean isAlternate) {
-     this.isAlternate = isAlternate;
- }
- @Override
  public int calcByteSize() {
      int byteSize = super.calcByteSize();

  – 2.7.4

From 3fbd1aa6f036a36195c7511e1187daf0d29ec1b5 Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Fri, 2 Jun 2017 21:44:19 +1000 Subject: [PATCH 03/27] Some initializations. Root must of course not be alternate.

src/moa/classifiers/trees/HATADWIN.java | 13 ++++++++++– 1 file changed, 11 insertions(+), 2 deletions(-)

diff –git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java index 59c8876..d91577a 100644 — a/src/moa/classifiers/trees/HATADWIN.java +++ b/src/moa/classifiers/trees/HATADWIN.java @@ -25,6 +25,7 @@ import java.util.Random; import moa.classifiers.bayes.NaiveBayes; import moa.classifiers.core.conditionaltests.InstanceConditionalTest; import moa.classifiers.core.driftdetection.ADWIN; +import moa.classifiers.trees.HoeffdingTree.LearningNode; import moa.core.DoubleVector; import moa.core.MiscUtils; import moa.core.Utils; @@ -216,10 +217,10 @@ public class HATADWIN extends HoeffdingTree { //if (this.isAlternateTree == false) { if (this.ErrorChange == true) {//&& this.alternateTree == null) { //Start a new alternative tree : learning node - this.alternateTree = ht.newLearningNode();

- //this.alternateTree.isAlternateTree = true; + this.alternateTree = ht.newLearningNode(true); // isAlternate is set to true ht.alternateTrees++; } // Check condition to replace tree + else if (this.alternateTree != null && ((NewNode) this.alternateTree).isNullError() == false) { if (this.getErrorWidth() > 300 && ((NewNode) this.alternateTree).getErrorWidth() > 300) { double oldErrorRate = this.getErrorEstimation(); @@ -482,6 +483,14 @@ public class HATADWIN extends HoeffdingTree {

```
protected int switchedAlternateTrees;
```

-
- protected LearningNode newLearningNode(boolean isAlternate) {
- `AdaLearningNode aln = new AdaLearningNode(new double[0]);`
- `aln.setAlternate(false);`
- `return aln;`
- }
-
- @Override protected LearningNode newLearningNode(double[] initialClassObservations) { // IDEA: to choose different learning nodes depending on predictionOption – 2.7.4

From 90dd0aa207e3da4d2abe8503885a2de0d46818db Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Fri, 2 Jun 2017 21:59:51 +1000 Subject: [PATCH 05/27] Add AttemptToSplit; This is where I will need to ensure fresh nodes

created on an alternate tree are also set to alternate.

src/moa/classifiers/trees/HATADWIN.java | 88 ++
1 file changed, 88 insertions(+)

diff –git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java index d91577a..f85856f 100644 — a/src/moa/classifiers/trees/HATADWIN.java +++ b/src/moa/classifiers/trees/HATADWIN.java @@ -19,13 +19,22 @@ */ package moa.classifiers.trees;

+import java.util.Arrays; +import java.util.HashSet; import java.util.LinkedList; import java.util.List; import java.util.Random; +import java.util.Set; + import moa.classifiers.bayes.NaiveBayes; +import moa.classifiers.core.AttributeSplitSuggestion; import moa.classifiers.core.conditionaltests.InstanceConditionalTest; import moa.classifiers.core.driftdetection.ADWIN; +import moa.classifiers.core.splitcriteria.SplitCriterion; +import moa.classifiers.trees.HoeffdingTree.ActiveLearningNode; import moa.classifiers.trees.HoeffdingTree.LearningNode; +import moa.classifiers.trees.HoeffdingTree.Node; +import moa.classifiers.trees.HoeffdingTree.SplitNode; import moa.core.DoubleVector;

import moa.core.MiscUtils; import moa.core.Utils; @@ -529,6 +538,85 @@ public class HATADWIN extends HoeffdingTree { }

```
@Override
```

- protected void attemptToSplit(ActiveLearningNode node, SplitNode parent,
- int parentIndex) {
- if (!node.observedClassDistributionIsPure()) {
- SplitCriterion splitCriterion = (SplitCriterion) getPreparedClassOption(this.split
- AttributeSplitSuggestion[] bestSplitSuggestions = node.getBestSplitSuggestions(spl
- Arrays.sort(bestSplitSuggestions);
- boolean shouldSplit = false;
- if (bestSplitSuggestions.length < 2) {
- shouldSplit = bestSplitSuggestions.length > 0;
- } else {
- double hoeffdingBound = computeHoeffdingBound(splitCriterion.getRangeOfMerit(n
- this.splitConfidenceOption.getValue(), node.getWeightSeen());
- AttributeSplitSuggestion bestSuggestion = bestSplitSuggestions[bestSplitSugges
- AttributeSplitSuggestion secondBestSuggestion = bestSplitSuggestions[bestSplit
- if ((bestSuggestion.merit - secondBestSuggestion.merit > hoeffdingBound)
- || (hoeffdingBound < this.tieThresholdOption.getValue())) {
- shouldSplit = true;
- }
- // }
- if ((this.removePoorAttsOption != null)
- && this.removePoorAttsOption.isSet()) {
- Set<Integer> poorAtts = new HashSet<Integer>();
- // scan 1 - add any poor to set
- for (int i = 0; i < bestSplitSuggestions.length; i++) {
- if (bestSplitSuggestions[i].splitTest != null) {
- int[] splitAtts = bestSplitSuggestions[i].splitTest.getAttsTestDep
- if (splitAtts.length == 1) {

```
                        if (bestSuggestion.merit
                                - bestSplitSuggestions[i].merit > hoeffdingBound) {
                            poorAtts.add(new Integer(splitAtts[0]));
                        }
                    }
                }
            }
            // scan 2 - remove good ones from set
            for (int i = 0; i < bestSplitSuggestions.length; i++) {
                if (bestSplitSuggestions[i].splitTest != null) {
                    int[] splitAtts = bestSplitSuggestions[i].splitTest.getAttsTestDep
                    if (splitAtts.length == 1) {
                        if (bestSuggestion.merit
                                - bestSplitSuggestions[i].merit < hoeffdingBound) {
                            poorAtts.remove(new Integer(splitAtts[0]));
                        }
                    }
                }
            }
            for (int poorAtt : poorAtts) {
                node.disableAttribute(poorAtt);
            }
        }
    }
    if (shouldSplit) {
        AttributeSplitSuggestion splitDecision = bestSplitSuggestions[bestSplitSuggest
        if (splitDecision.splitTest == null) {
            // preprune - null wins
            deactivateLearningNode(node, parent, parentIndex);
        } else {
            SplitNode newSplit = newSplitNode(splitDecision.splitTest,
```

```
                        node.getObservedClassDistribution(),splitDecision.numSplits() );
                for (int i = 0; i < splitDecision.numSplits(); i++) {
                    Node newChild = newLearningNode(splitDecision.resultingClassDistributi
                    newSplit.setChild(i, newChild);
                }
                this.activeLeafNodeCount--;
                this.decisionNodeCount++;
                this.activeLeafNodeCount += splitDecision.numSplits();
                if (parent == null) {
                    this.treeRoot = newSplit;
                } else {
                    parent.setChild(parentIndex, newSplit);
                }
            }
            // manage memory
            enforceTrackerLimit();
        }
    }
}
```

- @Override public double[] getVotesForInstance(Instance inst) { if (this.treeRoot != null) { FoundNode[] foundNodes = filterInstance-ToLeaves(inst, – 2.7.4

From 4ec2d16378069c25353a9086a90ccc01c49b8598 Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Fri, 2 Jun 2017 23:03:55 +1000 Subject: [PATCH 06/27] Splitting so that child nodes get alternate status of parent nodes

src/moa/classifiers/trees/HATADWIN.java | 6 ++++++ 1 file changed, 6 insertions(+)

diff –git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java index 12099e7..179d9ec 100644 — a/src/moa/classifiers/trees/HATADWIN.java +++ b/src/moa/classifiers/trees/HATADWIN.java @@ -658,6 +658,12 @@ public class HATADWIN extends HoeffdingTree { // So the filter will still add any nodes found deeper down to foundNodes // This looks like a bug.

- ```
  if (foundNode.node != null){
  ```
- ```
      if (((NewNode)foundNode.node).isAlternate()){
  ```
- ```
          System.err.println("Alternate is being used for prediction");
  ```
- ```
          System.exit(1);
  ```
- ```
      }
  ```
- ```
  }
  ```

```
 Node leafNode = foundNode.node;
 if (leafNode == null) {
```

– 2.7.4

From c58bf12838fc5e7c318ba030ec8ca72ec46f8e56 Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Fri, 2 Jun 2017 23:15:34 +1000 Subject: [PATCH 08/27] Turning off subtree substitution does not result in VFDT behaviour

This is because of the alternate trees voting. Even though they never get promoted because promotion is tured off, the performance profile does not match VFDT because of the alternate subtrees getting to vote.

src/moa/classifiers/trees/HATADWIN.java | 16 +++++++++——- 1 file changed, 9 insertions(+), 7 deletions(-)

diff –git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java index 179d9ec..ac9f026 100644 — a/src/moa/classifiers/trees/HATADWIN.java +++ b/src/moa/classifiers/trees/HATADWIN.java @@ -251,7 +251,9 @@ public class HATADWIN extends HoeffdingTree { //if (gNumAlts>0) fDelta=fDelta/gNumAlts; double fN = 1.0 / (((NewNode) this.alternateTree).getErrorWidth()) + 1.0 / (this.getErrorWidth()); double Bound = Math.sqrt(2.0 * oldErrorRate * (1.0 - oldErrorRate) * Math.log(2.0 / fDelta) * fN); - if (Bound < oldErrorRate - altErrorRate) { + if (Bound < oldErrorRate - altErrorRate + && this.subtreeDepth() < 0 +) { // Switch alternate tree ht.activeLeafNodeCount -= this.numberLeaves(); ht.activeLeafNodeCount += ((NewNode) this.alternateTree).numberLeaves(); @@ -658,12 +660,12 @@ public class HATADWIN extends HoeffdingTree { // So the filter will still add any nodes found deeper down to foundNodes // This looks like a bug.

- ```
  if (foundNode.node != null){
  ```
- ```
      if (((NewNode)foundNode.node).isAlternate()){
  ```
- ```
          System.err.println("Alternate is being used for prediction");
  ```
  ```

- ```
  System.exit(1);
  ```
- ```
  }
  ```
- ```
  }
  ```

+// if (foundNode.node != null){ +// if (((NewNode)foundNode.node).isAlternate()){ +// System.err.println("Alternate is being used for prediction"); +// System.exit(1); +// } +// }

```
Node leafNode = foundNode.node;
if (leafNode == null) {
```

– 2.7.4

From d1b0ecfec622f6a68d2aa5f35fa5e65a6f5f7ccd Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Fri, 2 Jun 2017 23:33:51 +1000 Subject: [PATCH 09/27] Now try allowing only mainline nodes to vote, and it breaks.

What is happening is that without alternate votes and without subtree substitution, VFDT is not approximated at all. If only mainline nodes are allowed to vote, error remains low until drift occurs and learning stops. If only alternates are allowed to vote, there is no learning until drift occurs and then it takes off.

The scenario of interest is the first one. There must be a recovery that approximates VFDT but there isn't.

Following drift, even though there is no subtree substitution, and alternates are not allowed to vote, the accuracy must recover. But it doesn't. Mainline nodes stop learning after a drift. But they shouldn't. Whether there is an alternate tree or not, they must keep learning and must recover at least as VFDT does.

EvaluatePrequential -l trees.HATADWIN -s (generators.monash.AbruptDriftGenerator -o 0.800001 -c -z 5 -n 5 -v 5 -r 1 -b 200000) -i 400000 -f 1000

src/moa/classifiers/trees/HATADWIN.java | 12 +++++++—— 1 file changed, 7 insertions(+), 5 deletions(-)

diff –git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java index ac9f026..a3a0631 100644 — a/src/moa/classifiers/trees/HATADWIN.java +++ b/src/moa/classifiers/trees/HATADWIN.java @@ -654,7 +654,9 @@ public class HATADWIN extends HoeffdingTree { DoubleVector result = new DoubleVector(); int predictionPaths = 0; for (FoundNode foundNode : foundNodes) { - if (foundNode.parentBranch != -999) { + if (foundNode.node != null){ + if(!((NewNode)foundNode.node).isAlternate()){ +// if (foundNode.parentBranch != -999) { // this only works one level down // Otherwise it doesn't - the node will just have a split index
```

as parent branch // So the filter will still add any nodes found deeper down to foundNodes @@ -681,11 +683,11 @@ public class HATADWIN extends HoeffdingTree { predictionPaths++; } } - - if(predictionPaths != 1) { - System.err.println("predictionPaths != 1"); - System.exit(1); } +// if(predictionPaths < 1) { +// System.err.println("predictionPaths = 0"); +// System.exit(1); +// }

```
//if (predictionPaths > this.maxPredictionPaths) {
// this.maxPredictionPaths++;
```

2.7.4

From
913b646fc033486551996369e61449eec65f95af
Mon
Sep
17
00:00:00
2001
From:
Chai-
tanya
Man-
apra-
gada
cman39@
student.
monash.
edu.
au
Date:
Fri,
2
Jun
2017
23:50:57
+1000
Sub-
ject:
[PATCH
10/27]
I
made
a
mis-
take
in
the
last
one,
but,
the
bug
re-
mains

Alternates
vote
as
soon
as
drift
oc-
curs
even
though
tree
sub-
sti-
tu-
tion
is
turned
off.
They
are
the
only
nodes
do-
ing
any
learn-
ing
af-
ter
drift-

turn
off

I had made a mistake in the last experiment with the braces, fixed now.

—

src/moa/classifiers/trees/HATADWIN.java | 19 ++++++++———— 1 file changed, 8 insertions(+), 11 deletions(-)

diff –git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java index a3a0631..405f7cb 100644 — a/src/moa/classifiers/trees/HATADWIN.java +++ b/src/moa/classifiers/trees/HATADWIN.java @@ -252,7 +252,7 @@ public class HATADWIN extends HoeffdingTree { double fN = 1.0 / (((NewNode) this.alternateTree).getErrorWidth()) + 1.0 / (this.getErrorWidth()); double Bound = Math.sqrt(2.0 * oldErrorRate * (1.0 - oldErrorRate) * Math.log(2.0 / fDelta) * fN); if (Bound < oldErrorRate - altErrorRate - && this.subtreeDepth() < 0 + && this.subtreeDepth() < 0) { // Switch alternate tree ht.activeLeafNodeCount -= this.numberLeaves(); @@ -654,20 +654,18 @@ public class HATADWIN extends HoeffdingTree { DoubleVector result = new DoubleVector(); int predictionPaths = 0; for (FoundNode foundNode : foundNodes) { - if (foundNode.node != null){ - if(!((NewNode)foundNode.node).isAlternate()){ // if (foundNode.parentBranch != -999) { // this only works one level down // Otherwise it doesn't - the node will just have a split index as parent branch // So the filter will still add any nodes found deeper down to foundNodes // This looks like a bug.

-// if (foundNode.node != null){ -// if (((NewNode)foundNode.node).isAlternate()){ -// System.err.println("Alternate is being used for prediction"); -// System.exit(1); -// } -// } + if (foundNode.node != null){ + if (((NewNode)foundNode.node).isAlternate()){ + System.err.println("Alternate is being used for prediction even though tree substitution is off"); + //System.exit(1); + } + }

```
Node leafNode = foundNode.node;
if (leafNode == null) {
```

@@ -682,8 +680,7 @@ public class HATADWIN extends HoeffdingTree { result.addValues(dist); predictionPaths++; } - } - } + // if(predictionPaths < 1) { // System.err.println("predictionPaths = 0"); // System.exit(1); – 2.7.4

From 1b77067da80bf01cfba258b9bae610e1b1599566 Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Sat, 3 Jun 2017 00:12:22 +1000 Subject: [PATCH 11/27] Alternates must not predict.

This would seem to be the point of the -999 test. Since it doesn't work, alternates are predicting.

But there is much more to this. As noted earlier, simply disabling alternates doesn't help.

What's happening is that the mainline does fine until drift happens. Then even if tree substitution is turned off, alternates predict. But merely turning off alternate prediction shouldn't be a problem.

Since it is a problem, it shows that the mainline is not behaving like VFDT. It stops learning the moment an alternate is created. Further tests show that it actually stops learning if the alternate is allowed to learn. Examples are filtering down to the alternates leaving the mainline dry.

These tests will follow now.

src/moa/classifiers/trees/HATADWIN.java | 12 +++++++—— 1 file changed, 6 insertions(+), 6 deletions(-)

diff –git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java index 405f7cb..ad24db4 100644 — a/src/moa/classifiers/trees/HATADWIN.java +++ b/src/moa/classifiers/trees/HATADWIN.java @@ -660,12 +660,12 @@ public class HATADWIN extends HoeffdingTree { // So the filter will still add any nodes found deeper down to foundNodes // This looks like a bug.

- ```
  if (foundNode.node != null){
  ```
- ```
      if (((NewNode)foundNode.node).isAlternate()){
  ```
- ```
          System.err.println("Alternate is being used for prediction even tho
  ```
- ```
          //System.exit(1);
  ```
- ```
      }
  ```
- ```
  }
  ```

+// if (foundNode.node != null){ +// if (((NewNode)foundNode.node).isAlternate()){ +// System.err.println("Alternate is being used for prediction even though tree substitution is off"); +// //System.exit(1); +// } +// }

```
                Node leafNode = foundNode.node;
                if (leafNode == null) {
```

– 2.7.4

From 8452e5ce7939d6bd8f8274e3c1ee72efd77bff4f Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Sat, 3 Jun 2017 00:14:08 +1000 Subject: [PATCH 12/27] Mainline learns if alternate doesn't

src/moa/classifiers/trees/HATADWIN.java | 70 ++++++++++++++++++————— —— 1 file changed, 26 insertions(+), 44 deletions(-)

diff –git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java index 3511fe2..6ed4541 100644 — a/src/moa/classifiers/trees/HATADWIN.java +++ b/src/moa/classifiers/trees/HATADWIN.java @@ -648,49 +648,31 @@ public class HATADWIN extends HoeffdingTree {

```
@Override
public double[] getVotesForInstance(Instance inst) {
```

- ```
  if (this.treeRoot != null) {
  ```
- ```
      FoundNode[] foundNodes = filterInstanceToLeaves(inst,
  ```
- ```
              null, -1, false);
  ```
- ```
      DoubleVector result = new DoubleVector();
  ```
- ```
      int predictionPaths = 0;
  ```
- ```
      for (FoundNode foundNode : foundNodes) {
  ```

-// if (foundNode.parentBranch != -999) {

- ```
              // this only works one level down
  ```
- ```
              // Otherwise it doesn't - the node will just have a split index as parent b
  ```
- ```
              // So the filter will still add any nodes found deeper down to foundNodes
  ```
- ```
              // This looks like a bug.
  ```
- -// if (foundNode.node != null){ -// if (((NewNode)foundNode.node).isAlternate()){ -// System.err.println(“Alternate is being used for prediction even though tree substitution is off”); -// //System.exit(1); -// } -// }
- ```
  
  ```
- ```
              Node leafNode = foundNode.node;
  ```
- ```
              if (leafNode == null) {
  ```
- ```
                  leafNode = foundNode.parent;
  ```
- ```
              }
  ```
  ```

```
            double[] dist = leafNode.getClassVotes(inst, this);
            //Albert: changed for weights
            //double distSum = Utils.sum(dist);
            //if (distSum > 0.0) {
            // Utils.normalize(dist, distSum);
            //}
            result.addValues(dist);
            predictionPaths++;
        }
-// if(predictionPaths < 1) { -// System.err.println("predictionPaths = 0"); -// System.exit(1); -// }

    //if (predictionPaths > this.maxPredictionPaths) {
    // this.maxPredictionPaths++;
    //}
    return result.getArrayRef();
 }
 return new double[0];
if (this.treeRoot != null) {
    FoundNode[] foundNodes = filterInstanceToLeaves(inst,
            null, -1, false);
    DoubleVector result = new DoubleVector();
    int predictionPaths = 0;
    for (FoundNode foundNode : foundNodes) {

        if (foundNode.node != null){
            if (!((NewNode)foundNode.node).isAlternate()){

                Node leafNode = foundNode.node;
                if (leafNode == null) {
                    leafNode = foundNode.parent;
```

```
                }
                double[] dist = leafNode.getClassVotes(inst, this);

                result.addValues(dist);
                predictionPaths++;

                return result.getArrayRef();
            }

        }
    }
}
return new double[0];
```

} } No newline at end of file – 2.7.4

From 073c2b4c4705a6f9cf8b7d29a1e97f62c0d779bb Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Sat, 3 Jun 2017 00:24:10 +1000 Subject: [PATCH 14/27] Now turn on alternate learning and mainline stops.

EvaluatePrequential -l trees.HATADWIN -s (generators.monash.AbruptDriftGenerator -o 0.800001 -c -z 5 -n 5 -v 5 -r 1 -b 200000) -i 400000 -f 1000

src/moa/classifiers/trees/HATADWIN.java | 2 +- 1 file changed, 1 insertion(+), 1 deletion(-)

diff –git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java index 6ed4541..7ee2bdb 100644 — a/src/moa/classifiers/trees/HATADWIN.java +++ b/src/moa/classifiers/trees/HATADWIN.java @@ -284,7 +284,7 @@ public class HATADWIN extends HoeffdingTree { //} //learnFromInstance alternate Tree and Child nodes if (this.alternateTree != null) { -// ((NewNode) this.alternateTree).learnFromInstance(weightedInst, ht, parent, parentBranch); + ((NewNode) this.alternateTree).learnFromInstance(weightedInst, ht, parent, parentBranch); } int childBranch = this.instanceChildIndex(inst); Node child = this.getChild(childBranch); – 2.7.4

From 70a305cb31a91b4434b4eccd5adb878c6cf2c8d9 Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Sat, 3

Jun 2017 01:29:51 +1000 Subject: [PATCH 15/27] More clarity- allowing only non-alternate votes

src/moa/classifiers/trees/HATADWIN.java | 16 ++++++++++++—- 1 file changed, 12 insertions(+), 4 deletions(-)

diff –git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java index fd45974..9f45447 100644 — a/src/moa/classifiers/trees/HATADWIN.java +++ b/src/moa/classifiers/trees/HATADWIN.java @@ -64,6 +64,8 @@ public class HATADWIN extends HoeffdingTree {

```
private static final long serialVersionUID = 1L;
```

- private static long numInstances = 0;
- @Override public String getPurposeString() { return “Hoeffding Adaptive Tree for evolving data streams that uses ADWIN to replace branches for new ones.”; @@ -220,8 +222,9 @@ public class HATADWIN extends HoeffdingTree { //Compute ClassPrediction using filterInstanceToLeaf //int ClassPrediction = Utils.maxIndex(filterInstanceToLeaf(inst, null, -1).node.getClassVotes(inst, ht)); int ClassPrediction = 0;
- `if (filterInstanceToLeaf(inst, parent, parentBranch).node != null) {`
- `ClassPrediction = Utils.maxIndex(filterInstanceToLeaf(inst, parent, parentBran`
- `Node leaf = filterInstanceToLeaf(inst, parent, parentBranch).node;`
- `if (leaf != null) {`
- `ClassPrediction = Utils.maxIndex(leaf.getClassVotes(inst, ht));`

  `}`

  `boolean blCorrect = (trueClass == ClassPrediction);`

  @@ -341,8 +344,7 @@ public class HATADWIN extends HoeffdingTree { } } if (this.alternateTree != null) {

- `((NewNode) this.alternateTree).filterInstanceToLeaves(inst, this, -999,`
- `foundNodes, updateSplitterCounts);`
- `((NewNode) this.alternateTree).filterInstanceToLeaves(inst, this, -999, foundN`
  `// the -999 used to launch this subtree filter becomes inutile immediately fol`
  `// the top node of the subtree. Only the immediate children of a split will se`
  `// So a foundnode created further down cannot be distinguished from the mainli`

  @@ -427,6 +429,11 @@ public class HATADWIN extends HoeffdingTree {

  ```
  @Override
  public void learnFromInstance(Instance inst, HATADWIN ht, SplitNode parent, int parent
  ```

- `if(!this.isAlternate()){`

- ```
  System.err.println(numInstances);
  ```
- ```
  }
  ```
- ```
   int trueClass = (int) inst.classValue();
   //New option vore
   int k = MiscUtils.poisson(1.0, this.classifierRandom);
  ```

  @@ -650,6 +657,7 @@ public class HATADWIN extends HoeffdingTree { @Override public double[] getVotesForInstance(Instance inst) { if (this.treeRoot != null) {

- ```
  numInstances++;
  FoundNode[] foundNodes = filterInstanceToLeaves(inst,
          null, -1, false);
  DoubleVector result = new DoubleVector();
  ```

  – 2.7.4

From da4f1def0fbed9dd596b793a1e6647fd7a1ad179 Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Sat, 3 Jun 2017 03:01:20 +1000 Subject: [PATCH 17/27] Disable alternates of alternates

src/moa/classifiers/trees/HATADWIN.java | 39 ++++++++++++++++++++++++++++++++++++++- 1 file changed, 38 insertions(+), 1 deletion(-)

diff –git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java index bfcb9df..6c7405c 100644 — a/src/moa/classifiers/trees/HATADWIN.java +++ b/src/moa/classifiers/trees/HATADWIN.java @@ -36,6 +36,7 @@ import moa.classifiers.trees.HoeffdingTree.ActiveLearningNode; import moa.classifiers.trees.HoeffdingTree.LearningNode; import moa.classifiers.trees.HoeffdingTree.Node; import moa.classifiers.trees.HoeffdingTree.SplitNode; +import moa.classifiers.trees.SubConceptTree.NewNode; import moa.core.DoubleVector; import moa.core.MiscUtils; import moa.core.Utils; @@ -101,6 +102,10 @@ public class HATADWIN extends HoeffdingTree {

```
public void setAlternate(boolean isAlternate);
```

- ```
  public boolean isRoot();
  ```
-
- ```
  public void setRoot(boolean isRoot);
  ```
- ```
  }
  ```

  public static class AdaSplitNode extends SplitNode implements NewNode { @@ -120,6 +125,8 @@ public class HATADWIN extends HoeffdingTree {

```
 private boolean isAlternate = false;
```

- ```
   private boolean isRoot = false;
  ```
- ```
  @Override
  public boolean isAlternate() {
  ```
  ```

```
    return this.isAlternate;
```

@@ -353,6 +360,17 @@ public class HATADWIN extends HoeffdingTree {

```
     }
 }
```

-
- `@Override`
- `public boolean isRoot() {`
- `    return this.isRoot;`
- `}`
-
- `@Override`
- `public void setRoot(boolean isRoot) {`
- `    this.isRoot = isRoot;`
-
- `}`

}

public static class AdaLearningNode extends LearningNodeNBAdaptive implements NewNode { @@ -369,6 +387,8 @@ public class HATADWIN extends HoeffdingTree {

```
 private boolean isAlternate = false;
```

- `private boolean isRoot = false;`
- `@Override`
  `public boolean isAlternate() {`
  `    return this.isAlternate;`

@@ -514,6 +534,17 @@ public class HATADWIN extends HoeffdingTree {

```
     foundNodes.add(new FoundNode(this, splitparent, parentBranch));
 }
```

-
- `@Override`
- `public boolean isRoot() {`
- `    return this.isRoot ;`
- `}`
-

- `@Override`
- `public void setRoot(boolean isRoot) {`
- `    this.isRoot = isRoot;`
-
- `}`

  }

  protected int alternateTrees; @@ -562,6 +593,7 @@ public class HATADWIN extends HoeffdingTree { public void trainOnInstanceImpl(Instance inst) { if (this.treeRoot == null) { this.treeRoot = newLearningNode(false); // root cannot be alternate

- `     ((NewNode) this.treeRoot).setRoot(true);`
  `     this.activeLeafNodeCount = 1;`
  `}`
  `((NewNode) this.treeRoot).learnFromInstance(inst, this, null, -1);`

  @@ -643,10 +675,15 @@ public class HATADWIN extends HoeffdingTree { this.activeLeafNodeCount–; this.decisionNodeCount++; this.activeLeafNodeCount += splitDecision.numSplits();

- `            if (parent == null) {`
- `            if (((NewNode)node).isRoot()) {`
- `               ((NewNode)newSplit).setRoot(true);`
  `                this.treeRoot = newSplit;`
- `                // What if I had an alternate at the root level? Parent would be null`
- `                // This should resolve clearly the difference between root and root's`
  `            } else {`
  `                parent.setChild(parentIndex, newSplit);`
- `                // but now... what happens when root's alternate is here? it's parent`
- `                // it must be attached to the root node... so let an alternate subtree`
  `            }`
  `        }`
  `        // manage memory`

  – 2.7.4

From 0f00948a2559e5275b1109ba7cd434cd24b0a21d Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Sat, 3 Jun 2017 14:06:06 +1000 Subject: [PATCH 19/27] Recovery after drift (ie in spite of alternate learning)

A number of changes over this commit and the last enable this.

First, resolving any confusion between root and root's alternate in the split function. The check used to be if parent was null. Since learnFromInstance would've had root's alternate in with null parent, the mainline tree would be lost in the splitting process without an alternate promotion ever happening. This would of course destroy prediction. It was the lack of clear distinction between mainline and alternate nodes that allowed the old code to work even if the alternates took over with no promotion.

This fix alone restarts learning after the drift. It doesn't yet fully approximate VFDT with promotion turned off, we'll get there one step at a time.

I also created a way for alternate subtrees to attach to the mainline. Only the point of attachment is recorded.Any node with no attachment doesn't have to carry information about the attachment it is forked from- only the top alternate node and it's corresponding mainline node need to.

This fixes the -999 check issue that only worked one level down. This fix is still commented out, but it will be uncommented in the next commit.

src/moa/classifiers/trees/HATADWIN.java | 41 ++—
– 1 file changed, 35 insertions(+), 6 deletions(-)

diff –git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java
index 6c7405c..c6bf3e6 100644 — a/src/moa/classifiers/trees/HATADWIN.java
+++ b/src/moa/classifiers/trees/HATADWIN.java @@ -36,7 +36,6 @@
import moa.classifiers.trees.HoeffdingTree.ActiveLearningNode; import moa.classifiers.trees.HoeffdingTree.LearningNode; import moa.classifiers.trees.HoeffdingTree.Node;
import moa.classifiers.trees.HoeffdingTree.SplitNode; -import moa.classifiers.trees.SubConceptTree.NewNode;
import moa.core.DoubleVector; import moa.core.MiscUtils; import moa.core.Utils;
@@ -106,6 +105,10 @@ public class HATADWIN extends HoeffdingTree {

```
public void setRoot(boolean isRoot);
```

- `public void setMainlineNode(AdaSplitNode parent);`
-
- `public AdaSplitNode getMainlineNode();`
- }

public static class AdaSplitNode extends SplitNode implements NewNode { @@ -127,6 +130,8 @@ public class HATADWIN extends HoeffdingTree {

```
 private boolean isRoot = false;
```

- `private AdaSplitNode mainlineNode = null; //null by default unless there is an attachme`
- `@Override`

```
public boolean isAlternate() {
    return this.isAlternate;
```

@@ -251,6 +256,7 @@ public class HATADWIN extends HoeffdingTree {

```
          //Start a new alternative tree : learning node
          this.alternateTree = ht.newLearningNode(true); // isAlternate is set to true
```

- ```
          ((NewNode)this.alternateTree).setMainlineNode(this);
          ht.alternateTrees++;
       } // Check condition to replace tree
```

@@ -371,6 +377,16 @@ public class HATADWIN extends HoeffdingTree { this.isRoot = isRoot;

```
}
```

-
- ```
  @Override
  ```
- ```
  public void setMainlineNode(AdaSplitNode mainlineNode) {
  ```
- ```
      this.mainlineNode  = mainlineNode;
  ```
- ```
  }
  ```
-
- ```
  @Override
  ```
- ```
  public AdaSplitNode getMainlineNode() {
  ```
- ```
      return this.mainlineNode;
  ```
- ```
  }
  ```

  }

  public static class AdaLearningNode extends LearningNodeNBAdaptive implements NewNode { @@ -389,6 +405,8 @@ public class HATADWIN extends HoeffdingTree {

  ```
  private boolean isRoot = false;
  ```

- ```
  private AdaSplitNode mainlineNode = null; //null by default unless there is an attachme
  ```
- ```
  @Override
  public boolean isAlternate() {
      return this.isAlternate;
  ```

  @@ -545,6 +563,16 @@ public class HATADWIN extends HoeffdingTree { this.isRoot = isRoot;

  ```
  }
  ```

- ```
  @Override
  ```

- `public void setMainlineNode(AdaSplitNode mainlineNode) {`
- `    this.mainlineNode  = mainlineNode;`
- `}`
-
- `@Override`
- `public AdaSplitNode getMainlineNode() {`
- `    return this.mainlineNode;`
- `}`
- `}`

  protected int alternateTrees; @@ -678,12 +706,13 @@ public class HATADWIN extends HoeffdingTree { if (((NewNode)node).isRoot()) { ((NewNode)newSplit).setRoot(true); this.treeRoot = newSplit;

- `                    // What if I had an alternate at the root level? Parent would be null`
- `                    // This should resolve clearly the difference between root and root's`
- `                } else {`
- `                }`
- `                else if (((NewNode)node).getMainlineNode() != null) { // if the node happe`
- `                   //((NewNode)node).getMainlineNode().alternateTree = newSplit;`
- `                }`
- `                else {`
  `                    parent.setChild(parentIndex, newSplit);`
- `                    // but now... what happens when root's alternate is here? it's parent`
- `                    // it must be attached to the root node... so let an alternate subtree`
- `                }`
  `            }`
  `            // manage memory`

  – 2.7.4

From b8a06d016976480166f7b18d1fae7cd4f9b847a3 Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Sat, 3 Jun 2017 14:33:57 +1000 Subject: [PATCH 20/27] Attachment points for alternate subtrees

As described in previous commit. These will come in very handy for a neat subtree promotion.

src/moa/classifiers/trees/HATADWIN.java | 4 ++– 1 file changed, 2 insertions(+), 2 deletions(-)

diff –git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java index c6bf3e6..1f1f712 100644 — a/src/moa/classifiers/trees/HATADWIN.java +++ b/src/moa/classifiers/trees/HATADWIN.java @@ -256,7 +256,7 @@ public class HATADWIN extends HoeffdingTree {

```
            //Start a new alternative tree : learning node
            this.alternateTree = ht.newLearningNode(true); // isAlternate is set to true
```

- ```
  ((NewNode)this.alternateTree).setMainlineNode(this);
  ```
- ```
  ((NewNode)this.alternateTree).setMainlineNode(this); // this node is the alter
  ```

```
            ht.alternateTrees++;
        } // Check condition to replace tree
```

@@ -708,7 +708,7 @@ public class HATADWIN extends HoeffdingTree { this.treeRoot = newSplit; } else if (((NewNode)node).getMainlineNode() != null) { // if the node happens to have a mainline attachment - //((NewNode)node).getMainlineNode().alternateTree = newSplit; + ((NewNode)node).getMainlineNode().alternateTree = newSplit; } else { parent.setChild(parentIndex, newSplit); – 2.7.4

From 6f62e63402bd856479228ea317662879e9e905ae Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Sat, 3 Jun 2017 15:01:14 +1000 Subject: [PATCH 21/27] Spurious weighting turned off

There was no mention of this in the paper. It is simply weighting everything and actually leads to worse performance than VFDT on this test.

Turning it off lets it match VFDT performance. Next, I will enable subtree promotion and get closer to HAT-ADWIN as described in the paper and debugged..

src/moa/classifiers/trees/HATADWIN.java | 8 ++++—- 1 file changed, 4 insertions(+), 4 deletions(-)

diff –git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java index 1f1f712..8e9b56c 100644 — a/src/moa/classifiers/trees/HATADWIN.java +++ b/src/moa/classifiers/trees/HATADWIN.java @@ -477,10 +477,10 @@ public class HATADWIN extends HoeffdingTree { //New option vore int k = MiscUtils.poisson(1.0, this.classifierRandom); Instance weightedInst = inst.copy(); - if (k > 0) { - weightedInst.setWeight(inst.weight() * k); - // this wasn't in the paper - } +// if (k > 0) { +// weightedInst.setWeight(inst.weight()

* k); +// // this wasn't in the paper +// } //Compute ClassPrediction using filterInstanceToLeaf int ClassPrediction = Utils.maxIndex(this.getClassVotes(inst, ht));

2.7.4

From 1d46111cd93475b914d5c4e3cb8213063ba0c5c1 Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Sat, 3 Jun 2017 15:25:58 +1000 Subject: [PATCH 22/27] Subtree substitution turned back on.

Also, the weighting has been left as is to compare with Bifet's version. The results actually match Bifet's HATADWIN. Why? Are the subtrees no deeper than two levels?

—

src/moa/classifiers/trees/HATADWIN.java | 91 ++++++++++++++++++++++++++++++—
– 1 file changed, 77 insertions(+), 14 deletions(-)

diff –git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java
index 8e9b56c..ed5e2c4 100644 — a/src/moa/classifiers/trees/HATADWIN.java
+++ b/src/moa/classifiers/trees/HATADWIN.java @@ -31,11 +31,6 @@ import
moa.classifiers.core.AttributeSplitSuggestion; import moa.classifiers.core.conditionaltests.InstanceConditionalTes

import moa.classifiers.core.driftdetection.ADWIN; import moa.classifiers.core.splitcriteria.SplitCriterion;
-import moa.classifiers.multilabel.trees.ISOUPTree.LeafNode; -import
moa.classifiers.trees.HoeffdingTree.ActiveLearningNode; -import moa.classifiers.trees.HoeffdingTree.LearningNod
-import moa.classifiers.trees.HoeffdingTree.Node; -import moa.classifiers.trees.HoeffdingTree.SplitNode;
import moa.core.DoubleVector; import moa.core.MiscUtils; import moa.core.Utils;
@@ -84,7 +79,9 @@ public class HATADWIN extends HoeffdingTree { //public
boolean getErrorChange(); public int numberLeaves();

- public double getErrorEstimation();
- void setAlternateStatusForSubtreeNodes(boolean isAlternate);
-
- public double getErrorEstimation();

public double getErrorWidth();

@@ -109,6 +106,10 @@ public class HATADWIN extends HoeffdingTree {

public AdaSplitNode getMainlineNode();

- public void setParent(AdaSplitNode parent);
-
- public AdaSplitNode getParent();
- }

public static class AdaSplitNode extends SplitNode implements NewNode
{ @@ -132,6 +133,20 @@ public class HATADWIN extends HoeffdingTree
{

private AdaSplitNode mainlineNode = null; //null by default unless there is an attachme

- private AdaSplitNode parent = null;
-
- @Override
- public void setParent(AdaSplitNode parent) {
- this.parent = parent;
-
- }
-
- @Override
- public AdaSplitNode getParent() {
- return this.parent;

- }
-
- @Override
  public boolean isAlternate() {
      return this.isAlternate;

  @@ -269,18 +284,33 @@ public class HATADWIN extends HoeffdingTree { double fN = 1.0 / (((NewNode) this.alternateTree).getErrorWidth()) + 1.0 / (this.getErrorWidth()); double Bound = Math.sqrt(2.0 * oldErrorRate * (1.0 - oldErrorRate) * Math.log(2.0 / fDelta) * fN); if (Bound < oldErrorRate - altErrorRate
- && this.subtreeDepth() < 0
- //&& this.subtreeDepth() < 0
) {
  // Switch alternate tree
  ht.activeLeafNodeCount -= this.numberLeaves();
  ht.activeLeafNodeCount += ((NewNode) this.alternateTree).numberLeaves(
- killTreeChilds(ht);
- if (parent != null) {
- this.killTreeChilds(ht);
- ((NewNode)this.alternateTree).setAlternateStatusForSubtreeNodes(false)
- ((NewNode)(this.alternateTree)).setMainlineNode(null);
-
- if (!this.isRoot()) {
- if(parent == null){
- System.err.println("Non-root node has null parent");
- StringBuilder out = new StringBuilder();
-
- //((AdaSplitNode)ht.treeRoot).describeSubtree(ht, out, 2);
- this.describeSubtree(ht, out, 2);
-
- //System.err.print(out);
- //System.exit(0);
- }
  parent.setChild(parentBranch, this.alternateTree);

```
                    ((NewNode)this.alternateTree).setParent(this.getParent());
                    //((AdaSplitNode) parent.getChild(parentBranch)).alternateTree = n
                } else {
                    // Switch root tree
                    ht.treeRoot = ((AdaSplitNode) ht.treeRoot).alternateTree;
                   ((NewNode)(this.alternateTree)).setRoot(true);
                    ht.treeRoot = this.alternateTree;
                }
                ht.switchedAlternateTrees++;
            } else if (Bound < altErrorRate - oldErrorRate) {
```

@@ -310,6 +340,20 @@ public class HATADWIN extends HoeffdingTree {
} }

```
@Override
 public void setAlternateStatusForSubtreeNodes(boolean isAlternate) {

   this.setAlternate(isAlternate);

   for (Node child : this.children) {
     if (child != null) {
       ((NewNode)child).setAlternateStatusForSubtreeNodes(isAlternate);
     }
   }
 }


 @Override
 public void killTreeChilds(HATADWIN ht) {
     for (Node child : this.children) {
```

@@ -407,6 +451,20 @@ public class HATADWIN extends HoeffdingTree {

```
private AdaSplitNode mainlineNode = null; //null by default unless there is an attachme
private AdaSplitNode parent = null;

@Override
public void setParent(AdaSplitNode parent) {
```

- `    this.parent = parent;`
-
- `}`
-
- `@Override`
- `public AdaSplitNode getParent() {`
- `    return this.parent;`
- `}`
-
- `@Override`
  `public boolean isAlternate() {`
  `    return this.isAlternate;`

  @@ -477,10 +535,10 @@ public class HATADWIN extends HoeffdingTree { //New option vore int k = MiscUtils.poisson(1.0, this.classifierRandom); Instance weightedInst = inst.copy(); -// if (k > 0) { -// weightedInst.setWeight(inst.weight() * k); -// // this wasn't in the paper -// }
- `     if (k > 0) {`
- `         weightedInst.setWeight(inst.weight() * k);`
- `         // this wasn't in the paper`
- `     }`
  `     //Compute ClassPrediction using filterInstanceToLeaf`
  `     int ClassPrediction = Utils.maxIndex(this.getClassVotes(inst, ht));`

@@ -573,6 +631,11 @@ public class HATADWIN extends HoeffdingTree { return this.mainlineNode; }

- `@Override`
- `public void setAlternateStatusForSubtreeNodes(boolean isAlternate) {`
- `    this.setAlternate(isAlternate);`
- `}`
- `}`

protected int alternateTrees;

2.7.4

From 34bff5cbfd785b2e7386635c3b9cd8fea1f91bf6 Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Sat, 3 Jun 2017 15:50:37 +1000 Subject: [PATCH 23/27] That is exactly it. The existing implementation appears to work because

for most test cases it is very rare to see a subtree of greater depth than 0. Where a subtree of depth 1 is created- as in the attached example- their accuracy and kappa temp diverge. (The original performs slightly better because the level 1 child nodes in the alternate don't see a parent branch of -999 and they end up voting when they shouldn't. . .)

EvaluatePrequential -l trees.HoeffdingAdaptiveTree -s (generators.monash.AbruptDriftGenerator -o 0.700002 -c -z 2 -r 1 -b 200000) -i 400000 -f 1000

src/moa/classifiers/trees/HATADWIN.java | 22 +++++++———————— 1 file changed, 7 insertions(+), 15 deletions(-)

diff –git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java index ed5e2c4..8c9c945 100644 — a/src/moa/classifiers/trees/HATADWIN.java +++ b/src/moa/classifiers/trees/HATADWIN.java @@ -293,17 +293,9 @@ public class HATADWIN extends HoeffdingTree { ((NewNode)this.alternateTree).setAlternateStatusForSubtreeNodes ((NewNode)(this.alternateTree)).setMainlineNode(null);

```
-                   if (!this.isRoot()) {
-                      if(parent == null){
-                          System.err.println("Non-root node has null parent");
-                          StringBuilder out = new StringBuilder();
-
-                          //((AdaSplitNode)ht.treeRoot).describeSubtree(ht, out, 2);
-                          this.describeSubtree(ht, out, 2);
-                   System.out.print(this.alternateTree.subtreeDepth() + " " + this.subtre
-                          //System.err.print(out);
-                          //System.exit(0);
-                      }
-                   if (!this.isRoot()) {
                        parent.setChild(parentBranch, this.alternateTree);
                        ((NewNode)this.alternateTree).setParent(this.getParent());
                        //((AdaSplitNode) parent.getChild(parentBranch)).alternateTree = n
```

@@ -525,11 +517,11 @@ public class HATADWIN extends HoeffdingTree {

```
@Override
public void learnFromInstance(Instance inst, HATADWIN ht, SplitNode parent, int parent
```

-
- `if(!this.isAlternate()){`
- `System.err.println(numInstances);`
- `// this shows mainline learning nodes stop learning once drift occurs`
- `}`

+// +// if(!this.isAlternate()){ +// System.err.println(numInstances); +// // this shows mainline learning nodes stop learning once drift occurs +// }

```
    int trueClass = (int) inst.classValue();
    //New option vore
```

– 2.7.4

From 280cfcb52fa2a041fc84f1d97b6fb788fef2c69f Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Mon, 5 Jun 2017 09:45:31 +1000 Subject: [PATCH 24/27] HATADWINOriginal for testing purposes

Now let's add the original source and compare.

Apart from the fact that alternate subtrees of depth are hard to come by, there is yet another reason the bug that alternate nodes vote due to a buggy parent check (-999) work.

Both the root and its alternate have parent "null". When the alternate is split, since it's parent is null, root is set to the alternate split node.

Alternate is set to root, and recovery happens.

Since root substitution happens with an alternate of level 0 in the debugged algorithm, what we have is the alternate is almost immediately better than the root. SO whether it is deliberately substituted or accidentally set to root, it happens within the same epoch keeping error the same.

EvaluatePrequential -l trees.HATADWIN -s (generators.monash.AbruptDriftGenerator -o 0.700002 -p -z 5 -n 5 -v 5 -r 1 -i 0.800001 -b 200000) -i 400000 -f 1000

src/moa/classifiers/trees/HATADWINOriginal.java | 514 ++++++++++++++++++++++++
1 file changed, 514 insertions(+) create mode 100644 src/moa/classifiers/trees/HATADWINOriginal.java

diff –git a/src/moa/classifiers/trees/HATADWINOriginal.java b/src/moa/classifiers/trees/HATADWINOriginal.
new file mode 100644 index 0000000..1673ee4 — /dev/null +++ b/src/moa/classifiers/trees/HATADWINOriginal
@@ -0,0 +1,514 @@ +/ + HoeffdingAdaptiveTree.java + * Copyright (C) 2008 University of Waikato, Hamilton, New Zealand + * @author Albert Bifet (abifet at cs dot waikato dot ac dot nz) + + This program is free software; you can redistribute it and/or modify + * it under the terms of the GNU General Public License as published by + * the Free Software Foundation; either version 3 of the License, or + * (at your option) any later version. + + This program is distributed in the hope that it will be useful, + * but WITHOUT ANY WARRANTY; without even the implied warranty of + * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the + * GNU General Public License for more details. + + You should have received a copy of the GNU General Public License + * along with this program. If not, see http://www.gnu.org/licenses/. + + / +package moa.classifiers.trees; + +import java.util.LinkedList; +import java.util.List; +import java.util.Random; +import moa.classifiers.bayes.NaiveBayes; +import moa.classifiers.core.conditionaltests.InstanceConditionalTest; +import moa.classifiers.core.driftdetection.ADWIN; +import moa.core.DoubleVector; +import moa.core.MiscUtils; +import moa.core.Utils; +import moa.streams.generators.monash.Node; + +import com.yahoo.labs.samoa.instances.Instance; + +/** + * Hoeffding Adaptive Tree for evolving data streams. +

This adaptive Hoeffding Tree uses ADWIN to monitor performance of + * branches on the tree and to replace them with new branches when their + * accuracy decreases if the new branches are more accurate.

- – See details in:
- – Adaptive Learning from Evolving Data Streams. Albert Bifet, Ricard Gavaldà.
- – IDA 2009
- –
- –
- – Same parameters as HoeffdingTreeNBAdaptive
- – -l : Leaf prediction to use: MajorityClass (MC), Naive Bayes (NB) or NaiveBayes
- – adaptive (NBAdaptive).
- –
- –
- – @author Albert Bifet (abifet at cs dot waikato dot ac dot nz)
- – @version $Revision: 7 $
- */ +public class HATADWINOriginal extends HoeffdingTree {

•
• private static final long serialVersionUID = 1L;
•
• @Override
• public String getPurposeString() {
• `return "Hoeffding Adaptive Tree for evolving data streams that uses ADWIN to replace b`
• }
•
• /* public MultiChoiceOption leafpredictionOption = new MultiChoiceOption(
• `"leafprediction", 'l', "Leaf prediction to use.", new String[]{`
• `"MC", "NB", "NBAdaptive"}, new String[]{`
• `"Majority class",`
• `"Naive Bayes",`
• `"Naive Bayes Adaptive"}, 2);*/`
•
• public interface NewNode {
•
• `// Change for adwin`
• `//public boolean getErrorChange();`
• `public int numberLeaves();`
•
• `public double getErrorEstimation();`
•
• `public double getErrorWidth();`
•
• `public boolean isNullError();`
•
• `public void killTreeChilds(HATADWINOriginal ht);`
•
• `public void learnFromInstance(Instance inst, HATADWINOriginal ht, SplitNode parent, in`

```
•
•  public void filterInstanceToLeaves(Instance inst, SplitNode myparent, int parentBranch
•          boolean updateSplitterCounts);
• }
•
• public static class AdaSplitNode extends SplitNode implements NewNode
  {
•
•  private static final long serialVersionUID = 1L;
•
•  protected Node alternateTree;
•
•  protected ADWIN estimationErrorWeight;
•  //public boolean isAlternateTree = false;
•
•  public boolean ErrorChange = false;
•
•  protected int randomSeed = 1;
•
•  protected Random classifierRandom;
•
•  //public boolean getErrorChange() {
•  //     return ErrorChange;
•  //}
•  @Override
•  public int calcByteSizeIncludingSubtree() {
•      int byteSize = calcByteSize();
•      if (alternateTree != null) {
•          byteSize += alternateTree.calcByteSizeIncludingSubtree();
•      }
•      if (estimationErrorWeight != null) {
```

```
            byteSize += estimationErrorWeight.measureByteSize();
        }
        for (Node child : this.children) {
            if (child != null) {
                byteSize += child.calcByteSizeIncludingSubtree();
            }
        }
        return byteSize;
    }

    public AdaSplitNode(InstanceConditionalTest splitTest,
            double[] classObservations, int size) {
        super(splitTest, classObservations, size);
        this.classifierRandom = new Random(this.randomSeed);
    }

    public AdaSplitNode(InstanceConditionalTest splitTest,
            double[] classObservations) {
        super(splitTest, classObservations);
        this.classifierRandom = new Random(this.randomSeed);
    }

    @Override
    public int numberLeaves() {
        int numLeaves = 0;
        for (Node child : this.children) {
            if (child != null) {
                numLeaves += ((NewNode) child).numberLeaves();
            }
        }
        return numLeaves;
```

```
}

@Override
public double getErrorEstimation() {
    return this.estimationErrorWeight.getEstimation();
}

@Override
public double getErrorWidth() {
    double w = 0.0;
    if (isNullError() == false) {
        w = this.estimationErrorWeight.getWidth();
    }
    return w;
}

@Override
public boolean isNullError() {
    return (this.estimationErrorWeight == null);
}

// SplitNodes can have alternative trees, but LearningNodes can't
// LearningNodes can split, but SplitNodes can't
// Parent nodes are allways SplitNodes
@Override
public void learnFromInstance(Instance inst, HATADWINOriginal ht, SplitNode parent, in
    int trueClass = (int) inst.classValue();
    //New option vore
    int k = MiscUtils.poisson(1.0, this.classifierRandom);
    Instance weightedInst = inst.copy();
    if (k > 0) {
```

```
            //weightedInst.setWeight(inst.weight() * k);
        }
        //Compute ClassPrediction using filterInstanceToLeaf
        //int ClassPrediction = Utils.maxIndex(filterInstanceToLeaf(inst, null, -1).node.g
        int ClassPrediction = 0;
        if (filterInstanceToLeaf(inst, parent, parentBranch).node != null) {
            ClassPrediction = Utils.maxIndex(filterInstanceToLeaf(inst, parent, parentBran
        }

        boolean blCorrect = (trueClass == ClassPrediction);

        if (this.estimationErrorWeight == null) {
            this.estimationErrorWeight = new ADWIN();
        }
        double oldError = this.getErrorEstimation();
        this.ErrorChange = this.estimationErrorWeight.setInput(blCorrect == true ? 0.0 : 1
        if (this.ErrorChange == true && oldError > this.getErrorEstimation()) {
            //if error is decreasing, don't do anything
            this.ErrorChange = false;
        }

        // Check condition to build a new alternate tree
        //if (this.isAlternateTree == false) {
        if (this.ErrorChange == true) {//&& this.alternateTree == null) {
            //Start a new alternative tree : learning node
            this.alternateTree = ht.newLearningNode();
            //this.alternateTree.isAlternateTree = true;
            ht.alternateTrees++;
        } // Check condition to replace tree
        else if (this.alternateTree != null && ((NewNode) this.alternateTree).isNullError(
            if (this.getErrorWidth() > 300 && ((NewNode) this.alternateTree).getErrorWidth
```

```
                double oldErrorRate = this.getErrorEstimation();
                double altErrorRate = ((NewNode) this.alternateTree).getErrorEstimation();
                double fDelta = .05;
                //if (gNumAlts>0) fDelta=fDelta/gNumAlts;
                double fN = 1.0 / (((NewNode) this.alternateTree).getErrorWidth()) + 1.0 /
                double Bound = Math.sqrt(2.0 * oldErrorRate * (1.0 - oldErrorRate) * Math.

                if (Bound < oldErrorRate - altErrorRate) {
                    // Switch alternate tree
                    ht.activeLeafNodeCount -= this.numberLeaves();
                    ht.activeLeafNodeCount += ((NewNode) this.alternateTree).numberLeaves(
                    killTreeChilds(ht);


                    if (parent != null) {
                        parent.setChild(parentBranch, this.alternateTree);
                        //((AdaSplitNode) parent.getChild(parentBranch)).alternateTree = n
                    } else {
                        // Switch root tree
                        ht.treeRoot = ((AdaSplitNode) ht.treeRoot).alternateTree;
                    }
                    ht.switchedAlternateTrees++;
                } else if (Bound < altErrorRate - oldErrorRate) {
                    // Erase alternate tree
                    if (this.alternateTree instanceof ActiveLearningNode) {
                        this.alternateTree = null;
                        //ht.activeLeafNodeCount--;
                    } else if (this.alternateTree instanceof InactiveLearningNode) {
                        this.alternateTree = null;
                        //ht.inactiveLeafNodeCount--;
                    } else {
```

```
                    ((AdaSplitNode) this.alternateTree).killTreeChilds(ht);
                }
                ht.prunedAlternateTrees++;
            }
        }
    }
    //}
    //learnFromInstance alternate Tree and Child nodes
    if (this.alternateTree != null) {
        ((NewNode) this.alternateTree).learnFromInstance(weightedInst, ht, parent, par
    }
    int childBranch = this.instanceChildIndex(inst);
    Node child = this.getChild(childBranch);
    if (child != null) {
        ((NewNode) child).learnFromInstance(weightedInst, ht, this, childBranch);
    }
}

@Override
public void killTreeChilds(HATADWINOriginal ht) {
    for (Node child : this.children) {
        if (child != null) {
            //Delete alternate tree if it exists
            if (child instanceof AdaSplitNode && ((AdaSplitNode) child).alternateTree
                ((NewNode) ((AdaSplitNode) child).alternateTree).killTreeChilds(ht);
                ht.prunedAlternateTrees++;
            }
            //Recursive delete of SplitNodes
            if (child instanceof AdaSplitNode) {
                ((NewNode) child).killTreeChilds(ht);
            }
```

```
            if (child instanceof ActiveLearningNode) {
                child = null;
                ht.activeLeafNodeCount--;
            } else if (child instanceof InactiveLearningNode) {
                child = null;
                ht.inactiveLeafNodeCount--;
            }
        }
    }
}

//New for option votes
//@Override
@Override
public void filterInstanceToLeaves(Instance inst, SplitNode myparent,
        int parentBranch, List<FoundNode> foundNodes,
        boolean updateSplitterCounts) {
    if (updateSplitterCounts) {
        this.observedClassDistribution.addToValue((int) inst.classValue(), inst.weight
    }
    int childIndex = instanceChildIndex(inst);
    if (childIndex >= 0) {
        Node child = getChild(childIndex);
        if (child != null) {
            ((NewNode) child).filterInstanceToLeaves(inst, this, childIndex,
                    foundNodes, updateSplitterCounts);
        } else {
            foundNodes.add(new FoundNode(null, this, childIndex));
        }
    }
    if (this.alternateTree != null) {
```

```
          ((NewNode) this.alternateTree).filterInstanceToLeaves(inst, this, -999,
                  foundNodes, updateSplitterCounts);
      }
  }
}

public static class AdaLearningNode extends LearningNodeNBAdaptive
implements NewNode {

 private static final long serialVersionUID = 1L;

 protected ADWIN estimationErrorWeight;

 public boolean ErrorChange = false;

 protected int randomSeed = 1;

 protected Random classifierRandom;

 @Override
 public int calcByteSize() {
     int byteSize = super.calcByteSize();
     if (estimationErrorWeight != null) {
         byteSize += estimationErrorWeight.measureByteSize();
     }
     return byteSize;
 }

 public AdaLearningNode(double[] initialClassObservations) {
     super(initialClassObservations);
     this.classifierRandom = new Random(this.randomSeed);
```

```
}

@Override
public int numberLeaves() {
    return 1;
}

@Override
public double getErrorEstimation() {
    if (this.estimationErrorWeight != null) {
        return this.estimationErrorWeight.getEstimation();
    } else {
        return 0;
    }
}

@Override
public double getErrorWidth() {
    return this.estimationErrorWeight.getWidth();
}

@Override
public boolean isNullError() {
    return (this.estimationErrorWeight == null);
}

@Override
public void killTreeChilds(HATADWINOriginal ht) {
}

@Override
```

```
public void learnFromInstance(Instance inst, HATADWINOriginal ht, SplitNode parent, in
    int trueClass = (int) inst.classValue();
    //New option vore
    int k = MiscUtils.poisson(1.0, this.classifierRandom);
    Instance weightedInst = inst.copy();
    if (k > 0) {
        weightedInst.setWeight(inst.weight() * k);
    }
    //Compute ClassPrediction using filterInstanceToLeaf
    int ClassPrediction = Utils.maxIndex(this.getClassVotes(inst, ht));

    boolean blCorrect = (trueClass == ClassPrediction);

    if (this.estimationErrorWeight == null) {
        this.estimationErrorWeight = new ADWIN();
    }
    double oldError = this.getErrorEstimation();
    this.ErrorChange = this.estimationErrorWeight.setInput(blCorrect == true ? 0.0 : 1
    if (this.ErrorChange == true && oldError > this.getErrorEstimation()) {
        this.ErrorChange = false;
    }

    //Update statistics
    learnFromInstance(weightedInst, ht);   //inst

    //Check for Split condition
    double weightSeen = this.getWeightSeen();
    if (weightSeen
            - this.getWeightSeenAtLastSplitEvaluation() >= ht.gracePeriodOption.getVal
        ht.attemptToSplit(this, parent,
                parentBranch);
```

```
            this.setWeightSeenAtLastSplitEvaluation(weightSeen);
        }


        //learnFromInstance alternate Tree and Child nodes
       /*if (this.alternateTree != null)  {
        this.alternateTree.learnFromInstance(inst,ht);
        }
        for (Node child : this.children) {
        if (child != null) {
        child.learnFromInstance(inst,ht);
        }
        }*/
    }

    @Override
    public double[] getClassVotes(Instance inst, HoeffdingTree ht) {
        double[] dist;
        int predictionOption = ((HATADWINOriginal) ht).leafpredictionOption.getChosenIndex
        if (predictionOption == 0) { //MC
            dist = this.observedClassDistribution.getArrayCopy();
        } else if (predictionOption == 1) { //NB
            dist = NaiveBayes.doNaiveBayesPrediction(inst,
                    this.observedClassDistribution, this.attributeObservers);
        } else { //NBAdaptive
            if (this.mcCorrectWeight > this.nbCorrectWeight) {
                dist = this.observedClassDistribution.getArrayCopy();
            } else {
                dist = NaiveBayes.doNaiveBayesPrediction(inst,
                        this.observedClassDistribution, this.attributeObservers);
            }
```

```
    }
    //New for option votes
    double distSum = Utils.sum(dist);
    if (distSum * this.getErrorEstimation() * this.getErrorEstimation() > 0.0) {
        Utils.normalize(dist, distSum * this.getErrorEstimation() * this.getErrorEstim
    }
    return dist;
 }

 //New for option votes
 @Override
 public void filterInstanceToLeaves(Instance inst,
         SplitNode splitparent, int parentBranch,
         List<FoundNode> foundNodes, boolean updateSplitterCounts) {
    foundNodes.add(new FoundNode(this, splitparent, parentBranch));
 }
}
```

```
protected int alternateTrees;

protected int prunedAlternateTrees;

protected int switchedAlternateTrees;

@Override
protected LearningNode newLearningNode(double[] initialClassObservations) {
 // IDEA: to choose different learning nodes depending on predictionOption
 return new AdaLearningNode(initialClassObservations);
}
```

```
@Override
protected SplitNode newSplitNode(InstanceConditionalTest splitTest,
     double[] classObservations, int size) {
 return new AdaSplitNode(splitTest, classObservations, size);
}

@Override
protected SplitNode newSplitNode(InstanceConditionalTest splitTest,
     double[] classObservations) {
 return new AdaSplitNode(splitTest, classObservations);
}

@Override
public void trainOnInstanceImpl(Instance inst) {
 if (this.treeRoot == null) {
     this.treeRoot = newLearningNode();
     this.activeLeafNodeCount = 1;
 }
 ((NewNode) this.treeRoot).learnFromInstance(inst, this, null, -1);
}

//New for options vote
public FoundNode[] filterInstanceToLeaves(Instance inst,
     SplitNode parent, int parentBranch, boolean updateSplitterCounts) {
 List<FoundNode> nodes = new LinkedList<FoundNode>();
 ((NewNode) this.treeRoot).filterInstanceToLeaves(inst, parent, parentBranch, nodes,
         updateSplitterCounts);
 return nodes.toArray(new FoundNode[nodes.size()]);
}

@Override
```

- public double[] getVotesForInstance(Instance inst) {
- ```
  if (this.treeRoot != null) {
      FoundNode[] foundNodes = filterInstanceToLeaves(inst,
              null, -1, false);
      DoubleVector result = new DoubleVector();
      int predictionPaths = 0;
      for (FoundNode foundNode : foundNodes) {
          if (foundNode.parentBranch != -999) {
              Node leafNode = foundNode.node;
              if (leafNode == null) {
                  leafNode = foundNode.parent;
              }
              double[] dist = leafNode.getClassVotes(inst, this);
              //Albert: changed for weights
              //double distSum = Utils.sum(dist);
              //if (distSum > 0.0) {
              // Utils.normalize(dist, distSum);
              //}
              result.addValues(dist);
              //predictionPaths++;
          }
      }
      //if (predictionPaths > this.maxPredictionPaths) {
      // this.maxPredictionPaths++;
      //}
      return result.getArrayRef();
  }
  return new double[0];
  ```
- } +} No newline at end of file – 2.7.4
  ```

From 17be5d365c11a0d70b2ed004c625409a3e809530 Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Mon, 5 Jun 2017 10:26:51 +1000 Subject: [PATCH 25/27] Proof of "null parent" bug

This is self-explanatory. With subtree substitution turned off, null parent confuses root and it's alternate. As long as an alternate is created at root, it gets substituted in even with subtree promotion turned off.

Attempt to split was taken directly from HoeffdingTree.java

```
                    System.err.println("Tree Root has already been split. So it must be the
                          + "Because parent is null");
```

EvaluatePrequential -l trees.HATADWINOriginal -s (generators.monash.AbruptDriftGenerator -o 0.800001 -c -z 5 -n 5 -v 5 -r 1 -i 0.800001 -b 200000) -i 400000 -f 1000

src/moa/classifiers/trees/HATADWINOriginal.java | 106 +++++++++++++++++++++++++-
1 file changed, 105 insertions(+), 1 deletion(-)

diff –git a/src/moa/classifiers/trees/HATADWINOriginal.java b/src/moa/classifiers/trees/HATADWINOriginal.j
index 1673ee4..4b6d4ba 100644 — a/src/moa/classifiers/trees/HATADWINOriginal.java
+++ b/src/moa/classifiers/trees/HATADWINOriginal.java @@ -19,12 +19,20 @@ */ package moa.classifiers.trees;

+import java.util.Arrays; +import java.util.HashSet; import java.util.LinkedList; import java.util.List; import java.util.Random; +import java.util.Set; + import moa.classifiers.bayes.NaiveBayes; +import moa.classifiers.core.AttributeSplitSuggestion; import moa.classifiers.core.conditionaltests.InstanceConditionalTest; import moa.classifiers.core.driftdetection.ADWIN; +import moa.classifiers.core.splitcriteria.SplitCriterion; +import moa.classifiers.trees.HoeffdingTree.ActiveLearningNode; +import moa.classifiers.trees.HoeffdingTree.SplitNode; import moa.core.DoubleVector; import moa.core.MiscUtils; import moa.core.Utils; @@ -55,6 +63,8 @@ public class HATADWINOriginal extends HoeffdingTree {

```
private static final long serialVersionUID = 1L;
```

- private static long numInstances = 0;
- @Override public String getPurposeString() { return "Hoeffding Adaptive Tree for evolving data streams that uses ADWIN to replace branches for new ones."; @@ -167,6 +177,9 @@ public class HATADWINOriginal extends HoeffdingTree { // SplitNodes can have alternative trees, but LearningNodes can't // LearningNodes can split, but SplitNodes can't // Parent nodes are allways SplitNodes
- `/* (non-Javadoc)`
- `* @see moa.classifiers.trees.HATADWINOriginal.NewNode#learnFromInstance(com.yahoo.lab`

- */
  @Override
  public void learnFromInstance(Instance inst, HATADWINOriginal ht, SplitNode parent, in
      int trueClass = (int) inst.classValue();

  @@ -212,7 +225,11 @@ public class HATADWINOriginal extends Hoeffd-
  ingTree { double fN = 1.0 / (((NewNode) this.alternateTree).getErrorWidth())
  + 1.0 / (this.getErrorWidth()); double Bound = Math.sqrt(2.0 * oldError-
  Rate * (1.0 - oldErrorRate) * Math.log(2.0 / fDelta) * fN);

- if (Bound < oldErrorRate - altErrorRate) {
- System.out.println(this.alternateTree.subtreeDepth() + " " + this.subtreeD
-
- if (Bound < oldErrorRate - altErrorRate
- && this.subtreeDepth() < 0
-) {
  // Switch alternate tree
  ht.activeLeafNodeCount -= this.numberLeaves();
  ht.activeLeafNodeCount += ((NewNode) this.alternateTree).numberLeaves(

  @@ -482,7 +499,94 @@ public class HATADWINOriginal extends Hoeffd-
  ingTree { }

  @Override

- protected void attemptToSplit(ActiveLearningNode node, SplitNode par-
  ent,
- int parentIndex) {
- if (!node.observedClassDistributionIsPure()) {
- SplitCriterion splitCriterion = (SplitCriterion) getPreparedClassOption(this.split
- AttributeSplitSuggestion[] bestSplitSuggestions = node.getBestSplitSuggestions(spl
- Arrays.sort(bestSplitSuggestions);
- boolean shouldSplit = false;
- if (bestSplitSuggestions.length < 2) {
- shouldSplit = bestSplitSuggestions.length > 0;
- } else {
- double hoeffdingBound = computeHoeffdingBound(splitCriterion.getRangeOfMerit(n
- this.splitConfidenceOption.getValue(), node.getWeightSeen());
- AttributeSplitSuggestion bestSuggestion = bestSplitSuggestions[bestSplitSugges

```
        AttributeSplitSuggestion secondBestSuggestion = bestSplitSuggestions[bestSplit
        if ((bestSuggestion.merit - secondBestSuggestion.merit > hoeffdingBound)
                || (hoeffdingBound < this.tieThresholdOption.getValue())) {
            shouldSplit = true;
        }
        // }
        if ((this.removePoorAttsOption != null)
                && this.removePoorAttsOption.isSet()) {
            Set<Integer> poorAtts = new HashSet<Integer>();
            // scan 1 - add any poor to set
            for (int i = 0; i < bestSplitSuggestions.length; i++) {
                if (bestSplitSuggestions[i].splitTest != null) {
                    int[] splitAtts = bestSplitSuggestions[i].splitTest.getAttsTestDep
                    if (splitAtts.length == 1) {
                        if (bestSuggestion.merit
                                - bestSplitSuggestions[i].merit > hoeffdingBound) {
                            poorAtts.add(new Integer(splitAtts[0]));
                        }
                    }
                }
            }
            // scan 2 - remove good ones from set
            for (int i = 0; i < bestSplitSuggestions.length; i++) {
                if (bestSplitSuggestions[i].splitTest != null) {
                    int[] splitAtts = bestSplitSuggestions[i].splitTest.getAttsTestDep
                    if (splitAtts.length == 1) {
                        if (bestSuggestion.merit
                                - bestSplitSuggestions[i].merit < hoeffdingBound) {
                            poorAtts.remove(new Integer(splitAtts[0]));
                        }
                    }
```

```
                }
            }
            for (int poorAtt : poorAtts) {
                node.disableAttribute(poorAtt);
            }
        }
    }
    if (shouldSplit) {
        AttributeSplitSuggestion splitDecision = bestSplitSuggestions[bestSplitSuggest
        if (splitDecision.splitTest == null) {
            // preprune - null wins
            deactivateLearningNode(node, parent, parentIndex);
        } else {
            SplitNode newSplit = newSplitNode(splitDecision.splitTest,
                    node.getObservedClassDistribution(),splitDecision.numSplits() );
            for (int i = 0; i < splitDecision.numSplits(); i++) {
                Node newChild = newLearningNode(splitDecision.resultingClassDistributi
                newSplit.setChild(i, newChild);
            }
            this.activeLeafNodeCount--;
            this.decisionNodeCount++;
            this.activeLeafNodeCount += splitDecision.numSplits();
            if (parent == null) {
               if(this.treeRoot.getClass() != AdaLearningNode.class){
                   System.err.println("Tree Root has already been split. So it must be
                           + "Because parent is null");
               }

                this.treeRoot = newSplit;
            } else {
                parent.setChild(parentIndex, newSplit);
```

- }
- }
- // manage memory
- enforceTrackerLimit();
- }
- }
- }
-
-
- @Override public double[] getVotesForInstance(Instance inst) {
- numInstances++;
- if (this.treeRoot != null) {
  FoundNode[] foundNodes = filterInstanceToLeaves(inst,
  null, -1, false);

– 2.7.4

From 444f6e033f1f376a2f9e3f5384768f3262e7f9a2 Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Sun, 11 Jun 2017 16:38:35 +1000 Subject: [PATCH 26/27] Turning back on subtree substitution in the original HATADWIN

To return it back to its original state...

src/moa/classifiers/trees/HATADWINOriginal.java | 1 - 1 file changed, 1 deletion(-)

diff –git a/src/moa/classifiers/trees/HATADWINOriginal.java b/src/moa/classifiers/trees/HATADWINOriginal.
index 4b6d4ba..5da44c0 100644 — a/src/moa/classifiers/trees/HATADWINOriginal.java +++ b/src/moa/classifiers/trees/HATADWINOriginal.java @@ -228,7 +228,6 @@ public class HATADWINOriginal extends HoeffdingTree { System.out.println(this.alternateTree.subtreeDepth() + " " + this.subtreeDepth() + " " + (parent==null));

if (Bound < oldErrorRate - altErrorRate

- && this.subtreeDepth() < 0
) {
  // Switch alternate tree
  ht.activeLeafNodeCount -= this.numberLeaves();

– 2.7.4

From 381d4016b73f984ad42390dd5e32889d3a97d51b Mon Sep 17 00:00:00 2001 From: Chaitanya Manapragada cman39@student.monash.edu.au Date: Sun, 11 Jun 2017 17:55:12 +1000 Subject: [PATCH 27/27] To avoid parent mixups, all nodes now store their parents

---
src/moa/classifiers/trees/HATADWIN.java | 39 ++++++++++++++++++++++++++++++++++++---
1 file changed, 29 insertions(+), 10 deletions(-)

diff – git a/src/moa/classifiers/trees/HATADWIN.java b/src/moa/classifiers/trees/HATADWIN.java index 8c9c945..daec5c7 100644 — a/src/moa/classifiers/trees/HATADWIN.java +++ b/src/moa/classifiers/trees/HATADWIN.java @@ -249,7 +249,7 @@ public class HATADWIN extends HoeffdingTree { //Compute ClassPrediction using filterInstanceToLeaf //int ClassPrediction = Utils.maxIndex(filterInstanceToLeaf(inst, null, -1).node.getClassVotes(inst, ht)); int ClassPrediction = 0; -

@@ -283,6 +284,18 @@ public class HATADWIN extends HoeffdingTree {
//if (gNumAlts>0) fDelta=fDelta/gNumAlts;
double fN = 1.0 / (((NewNode) this.alternateTree).getErrorWidth()) + 1.0 / (this.getErrorWidth());
double Bound = Math.sqrt(2.0 * oldErrorRate * (1.0 - oldErrorRate) * Math.log(2.0 / fDelta) * fN);

-
Sys-
tem.out.print(this.alternateTree.subtreeDepth()
+ "
" +
this.subtreeDepth());

```
if
(!this.isRoot())
{ -
par-
ent.setChild(parentBranch,
this.alternateTree);
+
this.getParent().setChild(parentBranch,
this.alternateTree);
+
((NewN-
ode)(this.alternateTree)).setRoot(false);
((NewN-
ode)this.alternateTree).setParent(this.getParent());
//((AdaS-
plitN-
ode)
par-
ent.getChild(parentBranch)).alternateTree
=
null;
}
else
{ //
Switch
root
tree
((NewN-
ode)(this.alternateTree)).setRoot(true);
+
((NewN-
ode)(this.alternateTree)).setParent(null);
ht.treeRoot
=
this.alternateTree;
} +
this.alternateTree
=
null;
ht.switchedAlternateTrees++;
}
else
if
(Bound
<
al-
tEr-
ror-
Rate
-
old-
Er-
ror-
Rate)
{ //
Erase
```

—

- ```
  else { //if the node is neither root nor an alternate, it must have a main
  ```
- ```
  ((NewNode)node).getParent().setChild(parentIndex, newSplit);
  ```
- ```
  ((NewNode)newSplit).setParent(((NewNode)node).getParent());
  ```

```
}
}
// manage memory
```

– 2.7.4
```