

Special Session on Random Forest and Ensembles

Maite Termenón¹

¹Computational Intelligence Group

2012 January 27

- Motivation
- Background
- Basic Streaming Random Forest Algorithm
- Extending the Algorithm to Handle Concept Drift
- Extending the Algorithm to Handle Random Blocks of Labels
- Implementation and Experimental Settings
- Results and Discussion
- Conclusions

Article to Present

22

IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 23, NO. 1, JANUARY 2011

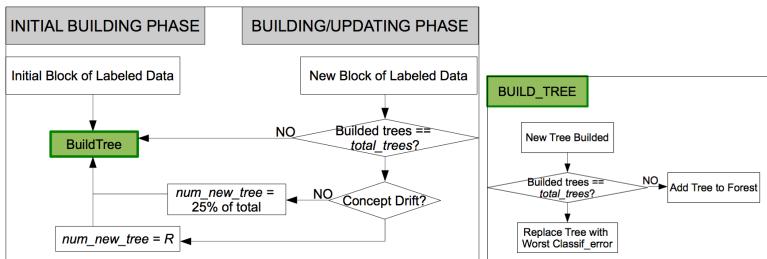
Classification Using Streaming Random Forests

Hanady Abdulsalam, David B. Skillicorn, *Member, IEEE*, and
Patrick Martin, *Member, IEEE Computer Society*

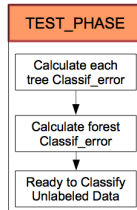
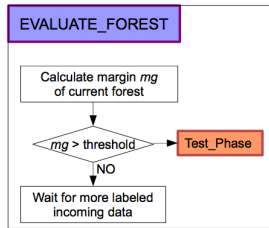
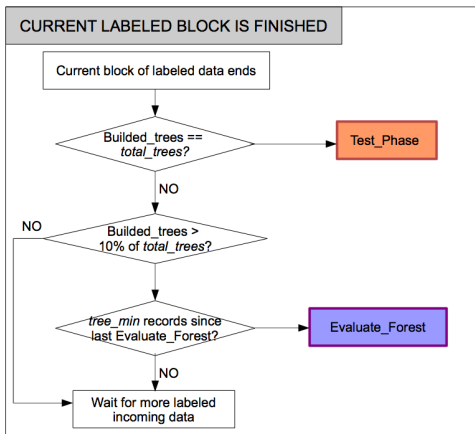
Summary Before Presentation

- Adapt RF algorithm to be able to classify with streaming data (multiclass and manage concept drift).
- Concepts: two window paradigm, margin function
- They present two different experiments: simulated and real data.
- Results show an adaptative algorithm, fast enough and low classification error rate.

Building Phase



Status Phase



Outline

- 1 Motivation
- 2 Background
- 3 Basic Streaming Random Forest Algorithm
- 4 Extending the Algorithm to Handle Concept Drift
- 5 Extending the Algorithm to Handle Random Blocks of Labeled Data
- 6 Implementation and Experimental Settings
- 7 Results and Discussion
- 8 Conclusions

Proposed Algorithm

- They introduce a stream classification algorithm:
 - Online,
 - running in amortized time, $\mathcal{O}(1)$,
 - able to handle intermittent arrival of labeled records,
 - able to adjust its parameters to respond to changing class boundaries (“concept drift”) in the data stream.

Proposed Algorithm

- When blocks of labeled data are short, algorithm is able to judge internally:
 - whether the quality of models updated from them is good enough for deployment on unlabeled records
 - whether further labeled records are required.
- Multiple target classes can be handled.

Data stream mining Algorithms

- They must be able to extract all necessary information from records with only one, or perhaps a few, passes over the data.
- It must be able to adapt the classification model to changes in the data stream, in particular to changes in the boundaries between classes (“concept drift”).
- Classification should be possible as soon as a sufficiently robust model has been built.
- Changes require a model update rather than a completely new model.

Classification Algorithms

- Consist of 3 phases:
 - 1 A training phase using labeled records.
 - 2 A test phase using previously unseen labeled records.
 - 3 A deployment phase that classifies unlabeled records.

Stream classification Algorithms

- Labeled and unlabeled records are mixed together in the stream.
- The training/test and deployment phases, therefore, must be interleaved.
- Stream classification of unlabeled records could be required:
 - At beginning of the stream, after some sufficiently long initial sequence of labeled records.
 - At specific moments in time.
 - For a specific block of records selected by an external analyst.

Possible scenarios

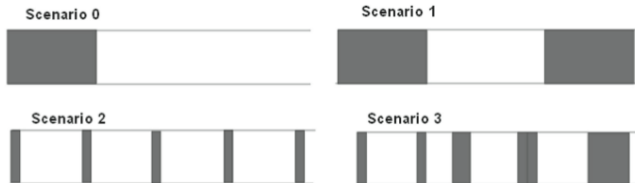


Figure: Possible scenarios for the positions of labeled records in streams (shaded areas represent labeled records).

Scenario 0

Scenario 0



- Basic Scenario.
- Do not permit class boundary changes.
- Fast enough to the arrival rates of records.

Scenario 1

Scenario 1



- Labeled data blocks are long enough to build or update the model.
- Boundaries between the classes in a newly arrived block may be different from previous blocks, so classification model must be updated.
- Updating process is straightforward, because the number of labeled records is sufficient to enable a complete update.

Scenario 2 and 3



- Not enough labeled records arrive at a time to guarantee that the classification model can be updated as necessary to reflect the changed class boundaries.
- Scenario 3 is the most realistic since the labeled blocks have random sizes and frequencies in the stream.

Scenario 2 and 3

- Model update should be able to:
 - alter the current model incrementally using any available labeled data.
 - guarantee that the new model is at least a better approximation than the old one
- If the model is unable to classify unlabeled records robustly, it should be able to detect this, and either not classify them, or label the classifications as potentially unreliable.

Main contribution

- Incremental stream classification algorithm:
 - It achieves high classification accuracy.
 - Even for multiclass classification problems.
- It combines ideas of:
 - Streaming decision trees
 - Random Forest

Algorithm Explanation

- 3 Phases:

- 1 Streaming decision tree construction is merged with Random Forests algorithm to formulate an incremental stream classification algorithm (Scenario 0).
- 2 How algorithm can be extended to handle concept drift in the stream (Scenario 1 using an entropy-based change-detection technique).
- 3 Algorithm is now able to decide whether the current model is ready for deployment or not (Scenarios 2 and 3).

Outline

- 1 Motivation
- 2 Background**
- 3 Basic Streaming Random Forest Algorithm
- 4 Extending the Algorithm to Handle Concept Drift
- 5 Extending the Algorithm to Handle Random Blocks of Labeled Data
- 6 Implementation and Experimental Settings
- 7 Results and Discussion
- 8 Conclusions

Background: Streaming Decision Trees

- Initially, a tree consists of a single frontier node.
- As each record arrives, it is routed down the tree, based on its attribute values and the inequalities of the internal nodes, until it reaches a frontier node.

Splitting Nodes

- Assume we have a general function G that checks an attribute's goodness for splitting (Information Gain or Gini index).
- At each frontier node, G is calculated for all attributes.
- The best and second best attributes are used to calculate:

$$\Delta G = G_{highest} - G_{second_highest}$$

- Algorithm recalculates G for all attributes as each new record arrives, or for every small number of data records (n_{min}), and updates ΔG continuously until it satisfies a stopping condition, $\Delta G > \epsilon$.

Hoeffding Bound

- It is used to decide when a frontier node has accumulated enough records for a robust decision about its attribute test to be made.

The Hoeffding bound states that, given a random variable r in the range L , and n independent values of r having mean \bar{r} , the true mean of r is at least $\bar{r} - \epsilon$, where

$$\epsilon = \sqrt{\frac{L^2 \ln(1/\delta)}{2n}}$$

with probability $1 - \delta$, and δ is a user-defined threshold.

Leaf Node

- A node is a leaf node only if the records that have arrived at this node, by the end of the training block, are all from the same class.
- This method of constructing streaming decision trees is able to only handle up to Scenario 1.

Two Window Paradigm

- To use entropy in data streams context, a two-window paradigm is typically used.
- Two sliding windows over the data stream are tracked:
 - one is the current window of data,
 - the other is a reference window that remembers the distribution of the stream.
- Algorithms for detecting concept changes compare the entropies of the current and reference windows.
- If the entropies differ by more than a defined amount then a change has occurred.

Background: Standard Random Forest Algorithm

- The Random Forests algorithm is an ensemble classification technique developed by Breiman in 2001.
- Random selection of records with replacement leaves some records (about a third of them) that are never used to build this tree.
- These records can be used to estimate the error rate of each tree.
- Random Forests classification error depends on:
 - Correlation among trees.
 - Strength of each individual tree.

Outline

- 1 Motivation
- 2 Background
- 3 Basic Streaming Random Forest Algorithm**
- 4 Extending the Algorithm to Handle Concept Drift
- 5 Extending the Algorithm to Handle Random Blocks of Labeled Data
- 6 Implementation and Experimental Settings
- 7 Results and Discussion
- 8 Conclusions

Selecting Blocks

- Standard RF repeatedly extracts subsets of records, and each one is used to build one of the trees of the ensemble.
- In a stream, blocks are selected from the incoming stream
- if the stream is sufficiently randomly ordered, this is equivalent to the standard case.
- In this basic streaming RF, block's size is fixed (*tree_window*).

Constructing the streaming RF

- Once a block of records has been selected, the RF tree-building algorithm is followed, with some refinements.
- Each newly arrived record is routed down the tree under construction until it reaches a frontier node.

Splitting Frontier Nodes

- When a frontier node has seen a block of records of user-defined size n_{min} , Hoeffding and Gini tests are applied.
- If Hoeffding bound test is satisfied:
 - Frontier node is transformed into an internal node with an inequality based on the best attribute and split point given by the Gini index tests.
 - The two children of this node become new frontier nodes.

Methodology

- If:
 - number of records reached by a frontier node exceeds a threshold, *node_window*, and
 - the node has not been split, and
 - the accumulated records are almost all from one class
- Then
 - Node is transformed into a leaf.
- If not:
 - Node is transformed to an internal node based on the best attribute and split point so far.

Basic Streaming RF vs standard RF

- The basic Streaming RF algorithm has classification accuracies approximately equal to those of the standard RF algorithm, but uses many fewer records for training than comparable stream classification algorithms.

Motivation

Background

Basic Streaming Random Forest Algorithm

Extending the Algorithm to Handle Concept Drift

Extending the Algorithm to Handle Random Blocks of Labeled

Implementation and Experimental Settings

Results and Discussion

Conclusions

Outline

- 1 Motivation
- 2 Background
- 3 Basic Streaming Random Forest Algorithm
- 4 Extending the Algorithm to Handle Concept Drift**
- 5 Extending the Algorithm to Handle Random Blocks of Labeled Data
- 6 Implementation and Experimental Settings
- 7 Results and Discussion
- 8 Conclusions

Extending the Algorithm to Handle Concept Drift

- They add features that enable the algorithm to tune its own parameters in response to the data that it sees.
- *tree_window* now changes depending on properties of the stream.
- Basic tree-building strategy is as before.

Classification Error

- After seen a $tree_{min}$ records: Classification error of each tree is calculated.
- Test set contains 10 percent of $tree_{min}$ unseen labeled records.
- If tree error $< tree_treshold$ then
 - current tree is complete.
- Else:
 - algorithm resumes building the current tree but with $tree_{min}$ set to half its previous value.
- Each tree consumes a different number of labeled records in its construction, never exceeding $2 \times tree_{min}$ records.
- n_{min} , $tree_{min}$, and $tree_treshold$ are assigned values initially.

Test Phase

- When required number of trees reached, calculate:
 - classification error of the entire forest,
 - classification error for each individual tree i .
- Individual errors ($tree_{error_i}$) are used to assign a weight to each tree's prediction.
- Algorithm derives new values for n_{min} , $tree_{min}$, and $tree_treshold$ to use in the next building phase, as shown next.

Next Building Phase

- $tree_threshold(t + 1) = \frac{1}{U} \sum_{i=1}^U tree_{error_i}(t)$, where U is the number of trees in the forest.
- $n_{min}(t + 1) = \frac{\ln(1/\delta)}{2(\overline{\Delta Gini(t)})^2}$, where

$$\Delta Gini(t) = Gini_{highest} - Gini_{second_highest}$$

is computed during the building phase at time t , and $\overline{\Delta Gini(t)}$ is the average value of $\Delta Gini(t)$.

- $tree_{min}(t + 1) = n_{min}(t + 1) * \overline{tree_{size}(t)}$, where $\overline{tree_{size}(t)}$ is the average of all tree sizes from the building phase at time t , measured in number of nodes per tree.

Next Building Phase

- A test is made to see if class boundaries have changed.
- If not:
 - Parameters from previous learning phases are used.
 - 25% of trees are replaced.
- Else:
 - Reset parameters.
 - Greater percentage of trees are changed.
- Replaced trees are those with largest values of $tree_{error}$.

Concept drift

- Three categories:
 - 1 Noise in data: no real change.
 - 2 Gradual change.
 - 3 Sudden change.

Concept drift

- Using an entropy-based change-detection technique, based on the two-windows paradigm.
- Difference in entropy for the current and reference windows is calculated for each attribute using counters to find the probabilities of occurrences for each value of the attribute.
- Differences are averaged to compute the average change in entropy.

Concept drift

- **Average entropy differences** (H) is normalized $[0, 1]$:
 $H' = H / \log_2(1/C)$, where C is the number of classes in the database?
 - It represents the % change in underlying concept.
- **Accumulated average of entropies differences**, H_{AVG} :
computed since the last recorded change.
- γ : constant threshold defined empirically.
- If $H' > \gamma + H_{AVG}$ then algorithm records a change in distribution.

Concept drift

$$R = \begin{cases} H' * U, & \text{if } (H' * U + \frac{1}{C}((1 - H')U) > U/2), \\ H' * U + U/2, & \text{otherwise,} \end{cases}$$

- If R is less than 25% of the number of trees U , then it is set to $R = R + 0.25 \times U$.
- If two concept drifts occur close to each other, it does not record a further change until it finishes its current update phase.

Concept drift

```
/*To update the forest at time  $t+1$ */  
compute normalized entropy ( $H'$ )  
compute  $H_{AVG}$   
if  $H' > \gamma + H_{AVG}$   
    call BuildTree to replace the least  
    accurate  $w\%$  of trees using  
     $tree_{threshold}(t+1)$ ,  $n_{min}(t+1)$ ,  
    and  $tree_{min}(t+1)$   
else  
    record a change in distribution  
    reset  $tree_{threshold}$ ,  $n_{min}$ , and  
     $tree_{min}$  to initial values  
    calculate  $R$   
    call BuildTree to replace the least  
    accurate  $R$  trees  
end if
```

Motivation

Background

Basic Streaming Random Forest Algorithm

Extending the Algorithm to Handle Concept Drift

Extending the Algorithm to Handle Random Blocks of Labeled

Implementation and Experimental Settings

Results and Discussion

Conclusions

The Margin Function

Initial Forest-Building Phase

Deciding when the Forest is Good Enough

Subsequent Building Phase

Time Complexity

Outline

- 1 Motivation
- 2 Background
- 3 Basic Streaming Random Forest Algorithm
- 4 Extending the Algorithm to Handle Concept Drift
- 5 Extending the Algorithm to Handle Random Blocks of Labeled Data**
- 6 Implementation and Experimental Settings
- 7 Results and Discussion
- 8 Conclusions

Random Blocks of Labeled Data

- Algorithm able to update its model using blocks of labeled records of any length.
- Determine if:
 - resulting classifier is “good enough” to be deployed, or
 - the algorithm should wait for further labeled records and use them to improve its quality first.
- This evaluation must be made whenever a block of labeled records ends.
- They call it, the **Evaluation Phase**.

Evaluation Phase

- It is based on the **margin function** (mg) defined by Breiman.
- A set of labeled records, *evaluation_set*, is kept from the labeled records encountered since the preceding evaluation phase.
- One percent of the labeled records are stored, calling them *eval_cnt*.

Marging Function

- Defined to measure the strength of an ensemble of classifiers during construction.

Given a Random Forest with U trees

$tree_1, tree_2, \dots, tree_U$, the margin function is defined to be

$$mg(X, Y) = avg_U I(tree_u(X) = Y) \\ - max_{j \neq Y} avg_U I(tree_u(X) = j),$$

where $1 \leq u \leq U$, $I(\cdot)$ is the indicator function, X is a labeled record from class Y , and $tree_u(X)$ is the class voted for by $tree_u$ for record X .

Initial Forest-Building Phase

- Algorithm attempts to grow the defined number of trees based on the initial block of labeled records in the stream.
- If the number of labeled records in the initial block is not sufficient to completely build the required forest:
 - Algorithm enters an evaluation phase, if enough trees were grown for evaluation.
 - Threshold = 10% of the required trees.
- Average of margin values is calculated (ignoring trees partially constructed):

$$\overline{mg} = \frac{1}{eval\ cnt} \sum_{i=1}^{eval\ cnt} mg_i(X_{C\ evaluation\ set}, Y).$$

Initial Forest-Building Phase

- If $\bar{m}g \geq mg_{threshold}$ then
 - It is deployed to classify unlabeled records.
 - Evaluation set is cleared.
 - Enters a test phase (to measure what classification error would be).
 - Starts collecting new evaluation records when incoming labeled records appear.
- else
 - no classification of unlabeled records takes place.
 - algorithm waits for the next block of labeled records to resume building the forest.

mg threshold value?

- A simple way:

$$mg_{threshold} = U/C$$

- It is the number of correct votes for a class if the forest was to vote uniformly randomly.

Algorithm confidence

- Positive values of $\bar{m}g$ define how confident the forest is in its classification.
- Algorithm confidence will be used to decide when to deploy the ensemble on unlabeled records:

$$conf = \begin{cases} \bar{m}g/U, & \text{if } \bar{m}g \geq 0; \\ 0, & \text{otherwise.} \end{cases}$$

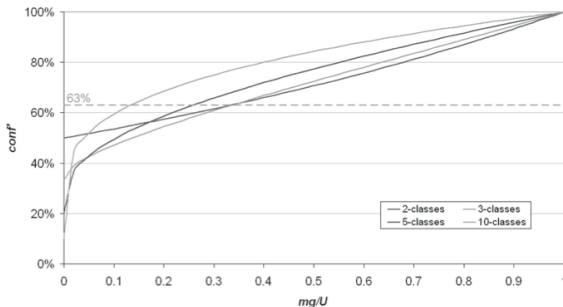
Another definition of $conf$

- The forest confidence might be greater than what a linear relation suggests:

$$conf' = \begin{cases} \left(\frac{1}{C}\right)^{\left(1 - \frac{\overline{mg}}{C-1}\right)}, & \text{if } \overline{mg} \geq 0; \\ 0, & \text{otherwise.} \end{cases}$$

Example

- Plot of $conf'$ versus positive values of $m\bar{g}/U$ for $U = 50$ trees and number of classes $C = 2, 3, 5, 10$.



Another definition of mg threshold

- Another definition of $mg_{threshold}$ is required when using $conf'$.
- They derive $mg'_{threshold}$ from the equation of $conf'$, given the accepted confidence level that is expected from the algorithm, $conf'_{given}$.

$$mg'_{threshold} = U * (1 - \log_{\frac{1}{c}} conf'_{given})^{C-1}$$

Subsequent Building Phase

- Three conditions may occur when a new block of labeled records is encountered:
 - Total number of trees less than required for the forest:
 - Target number of trees = remaining number of trees to complete de forest the first time.
 - Required number of trees is grown and no concept drift:
 - Target number of trees = 25% of total.
 - Required number of trees is grown and there is concept drift:
 - Target number of trees = R .

Subsequent Building Phase

- Algorithm grows the target number of trees one by one.
- Each time it completes a tree:
 - adds the tree to the forest if the total number of trees has not yet been reached, or
 - it replaces the existing tree with the worst classification performance by the new tree.

Current block of labeled data ends

```

/*building/updating phase*/
while more trees to build
  call BuildTree
  if block of labelled records ends
    if current number of trees  $\geq$  10% of total
      number of trees and  $eval\ cnt \geq 0.01 * tree_{min}$ 
      call EvaluateForest to calculate  $\overline{mg}$ 
      if  $\overline{mg} \geq (mg_{threshold} \text{ or } mg'_{threshold})$ 
        ignore tree under construction in case of
        initial building
        or restore old tree in case of update
        use current forest for classification
        reset evaluation set
      end if
    end if
    continue building trees when a new block of
    labelled records arrives
  end if
  if concept drift is detected
    reset evaluation set
  end if
end while

```


Time Complexity

- The time the algorithm takes to complete a building phase depends on:
 - 1 Time to build each tree.
 - 2 Time to evaluate the forest.
 - 3 Time to test the forest.

Time to Build each Tree

- Sum of the **times for reading** the training records,
- **passing them down trees** to the nodes,
- incrementing the counters,
- performing Hoeffding and Gini tests every n_{min} records,
- performing the entropy test for every entropy window of records, and
- **testing the tree.**

Time to Build each Tree

- Each tree is trained from a maximum of $2 \times tree_{min}$ number of records.
- Each tree is tested using a maximum of $0.1 \times 2 \times tree_{min}$ records.
- Assuming the trees are balanced, during training and testing, a record needs to pass through a maximum of $\log_2 tree_{size} + 1$ levels to a frontier node.
- So,

time to build a tree is therefore $(2 * tree_{min} + 0.1 * 2 * tree_{min}) * (\log_2 tree_{size} + 1)$, which gives a complexity of $\mathcal{O}(U * tree_{min} * \log_2 tree_{size})$ for building U trees.

Time to Evaluate the Forest

- Depends on:
 - reading the records of the evaluation set, and
 - passing them down all the trees of the current forest.
- Since the evaluation set contains a maximum of one percent of the number of records read, the time for evaluating the forest under construction is:

$U * 0.01 * 2 * tree_{min} * (\log_2 \overline{tree_{size}} + 1)$, which reduces to a complexity of $\mathcal{O}(U * tree_{min} * \log_2 \overline{tree_{size}})$.

Time to Test the Forest

- It depends on reading the test records and passing them down all the trees.
- Number of test records is constant.
- Time complexity for testing the forest under construction is:

$$\mathcal{O}(U * \log_2 \overline{tree_{size}})$$

Outline

- 1 Motivation
- 2 Background
- 3 Basic Streaming Random Forest Algorithm
- 4 Extending the Algorithm to Handle Concept Drift
- 5 Extending the Algorithm to Handle Random Blocks of Labeled Data
- 6 Implementation and Experimental Settings**
- 7 Results and Discussion
- 8 Conclusions

Testing Criteria

- Two main criteria:
 - The way of adapting in the case of concept drifts.
 - The decisions that the algorithm takes when a block of labeled data is not long enough to complete a training phase.
- They randomly choose a number of *stop points* to simulate the presence of blocks of labeled records in the stream.
- Each stop point defines the end of a block of labeled records and the start of the next block of labeled records.

Measuring the behavior of the algorithm

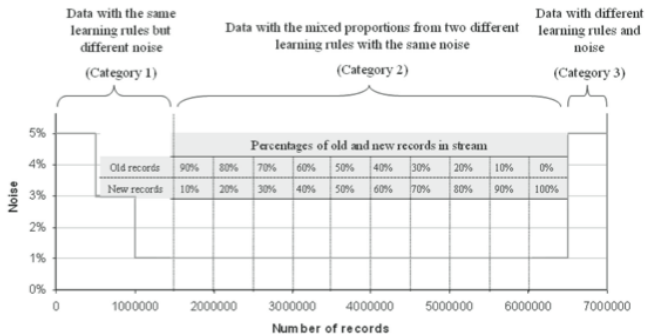
- 1 Forest classification errors when all training blocks are big enough to build/update the forest.
- 2 R , each time the algorithm enters an update phase.
- 3 Forest classification errors when blocks might not be big enough to complete build/update phases.
- 4 $\bar{m}g$, $mg_{threshold}$ and $mg'_{threshold}$, each time the forest is considered for evaluation ($conf'_{given} = 63\%$).
- 5 Classification errors for records of the evaluation sets.
- 6 Confidence values, $conf$ and $conf'$, each time the forest is evaluated.

Data Sets

- They test the algorithm on synthetic and real data sets.
- They use 50 stop points for the synthetic data set, 15 for one real data set and 10 for the other real data set.

Synthetic Data

- Generated using the DataGen data generation tool by Melli (<http://www.datasetgenerator.com/>).



Real Data

- They two real astronomical data sets, extracted from the Sloan Digital Sky Survey (SDSS - <http://www.sdss.org/>).
- SDSS offers astronomical data about more than one quarter of the sky, by capturing optical images.
- Experiments are based on the sixth release of SDSS, DR6.
- The maximum recorded uncertainty for the attributes is about two percent.

Two Subsets - PhotoObj

- PhotoObj:
 - 7 Attributes:
 - five attributes ($u, g, r, i,$ and z) that represent original color measurements in the photometric color system *ugriz*.
 - Two other attributes that represent the right ascension (RA), and the declination (DEC) of each object.
 - Two Classes: galaxies class and dwarf-stars class.
 - Data Set contains:
 - 500,000 records of class 1
 - 41,116 records of class 2

Two Subsets - PhotoObj

- PhotoObj:
 - They merge the data sets into one data set with balanced class ratios.
 - Randomly merging the two data sets by selecting records from class 1 with probability 0.9 and records from class 2 with probability 0.1.
 - Then, they replicate the records from class 2 ten times.
 - Resulting data set contains 782,730 records, with 371,517 records (48 percent) of class 1, and 411,160 records (52 percent) of class 2.

Two Subsets - SpecPhoto

- SpecPhoto:
 - 7 Atributes:
 - same five color attributes corrected to include the effect of redshift astronomical measure (dered_u,dered_g,dered_r,dered_i, and dered_z).
 - same two other attributes RA and DEC of each object, that also contains redshift measure.
 - Two Classes: galaxies=3 and stars=6.
 - Data Set contains:
 - 65,261 records of class 1
 - 20,441 records of class 2

Two Subsets - SpecPhoto

- SpecPhoto:
 - They replicate the records from class 2 three times:
 - data set with 51.7% of the class 1 and 48.3% of class 2.
 - They noticed that data records in the lower range of DEC values are not frequent enough to train the classifier after the concept drift is detected.
 - So, they replicate the whole block of data three times to produce a final data set with 158,128 records.

Outline

- 1 Motivation
- 2 Background
- 3 Basic Streaming Random Forest Algorithm
- 4 Extending the Algorithm to Handle Concept Drift
- 5 Extending the Algorithm to Handle Random Blocks of Labeled Data
- 6 Implementation and Experimental Settings
- 7 Results and Discussion**
- 8 Conclusions

Classification Accuracy for Scenario 1

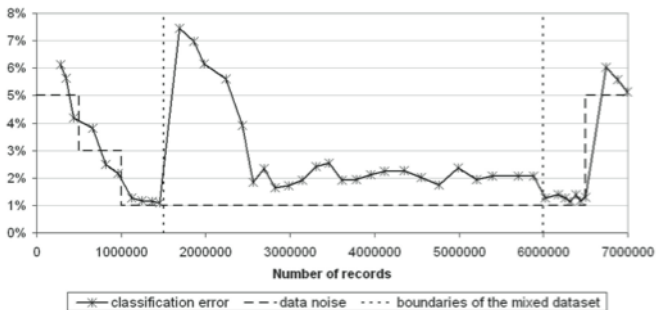


Figure: Classification errors as the algorithm adapts to concepts drifts

Comparison with Other Stream Classification Algorithms

- Comparison with decision tree-based stream classification algorithms:
 - Compare to extensions of streaming decision tree algorithm VFDT: CVFDT and sVFDT.
 - Conclusions:
 - Decision tree algorithms require much more data than our algorithm.
 - The use of the random forests construction for each individual tree improve on previous decision tree construction algorithms.
 - These algorithms can only handle concept drift by discarding the current model and learning a new one.

Comparison with Other Stream Classification Algorithms

- Comparison with ensemble-based stream classification algorithms:
 - They use a variety of component conventional classification algorithms in building their predictors.
 - Have only been tested on two-class problems.
 - We compare the percent noise and the percent expected error for [11], [12], [13], [14], [15].
 - Conclusions:
 - Only [13] achieves a prediction error rate comparable to the best achievable, only on a two-class problem.
 - Our algorithm almost always reaches the expected error rate, even for multiclass.

Dynamic Adjustment of R

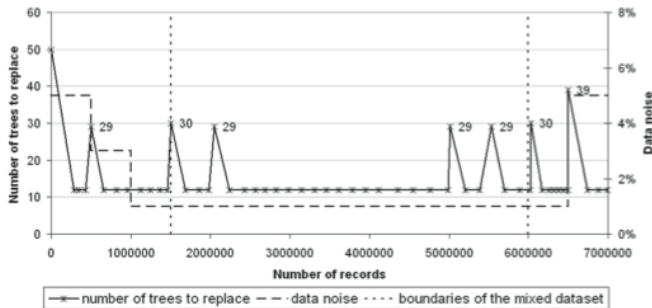


Figure: Number of trees to replace.

Margin Function Test - Synthetic

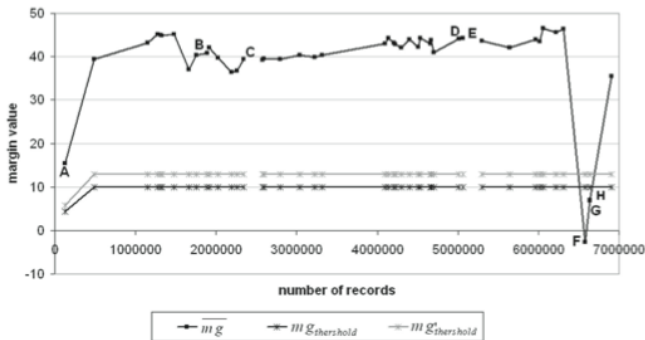


Figure: Values of \bar{m}_g , $m_g^{threshold}$, and $m_g'^{threshold}$ for synthetic data.

Margin Function Test - Real

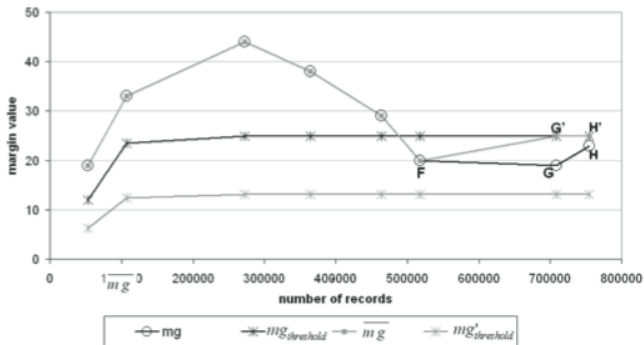


Figure: Values of $\bar{m}g$, $\bar{m}g'$, $mg_{threshold}$, and $mg'_{threshold}$ for PhotoObj.

Errors for Eval Sets - Synthetic

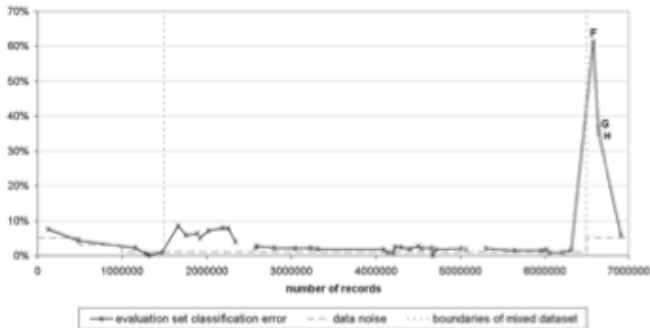


Figure: Evaluation set classification errors for synthetic data.

Errors for Eval Sets - Real

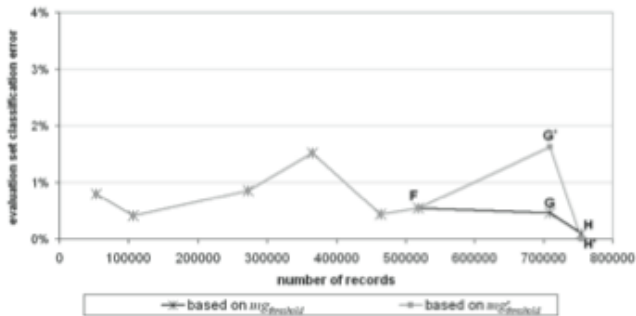


Figure: Evaluation set classification errors for PhotoObj.

Algorithm Confidence - Synthetic

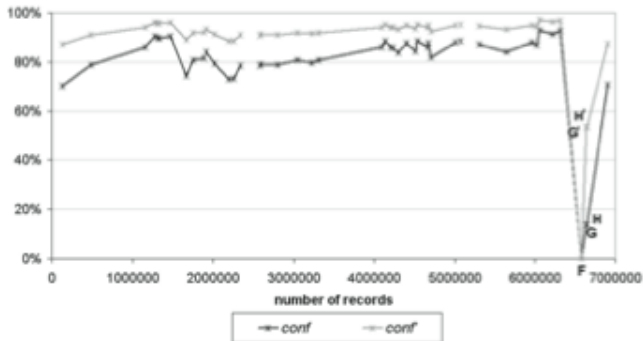


Figure: Confidence values for synthetic data.

Algorithm Confidence - Real

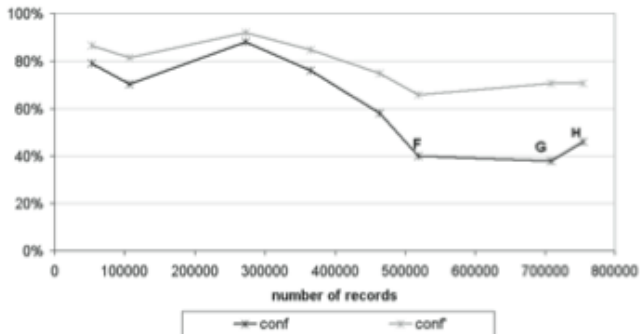


Figure: Confidence values for PhotoObj.

Classification Accuracy of the Forest - Synthetic

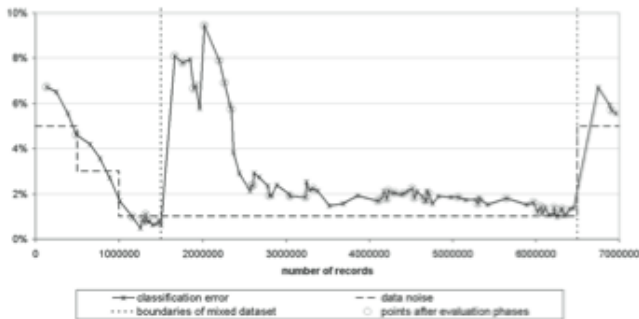


Figure: Forest classification errors for synthetic data.

Classification Accuracy of the Forest - PhotoObj

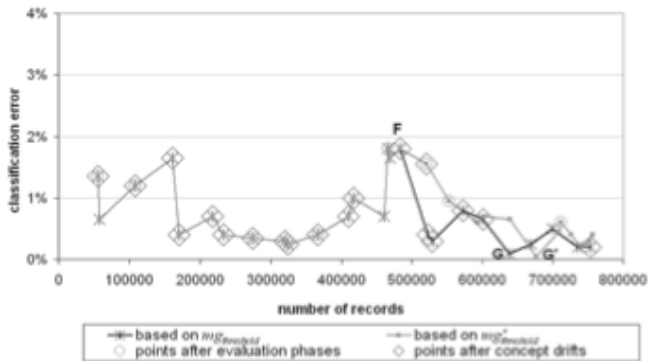


Figure: Forest classification errors for PhotoObj.

Classification Accuracy of standard Random Forest

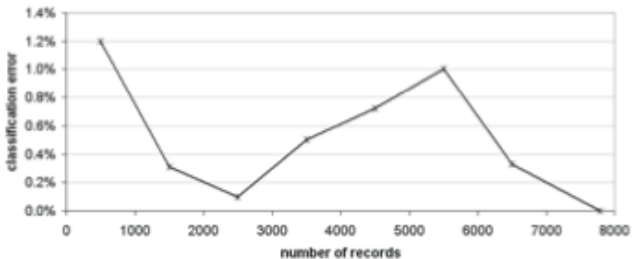


Figure: Classification errors of the Random Forests for PhotoObj.

Entropy values for SpecPhoto

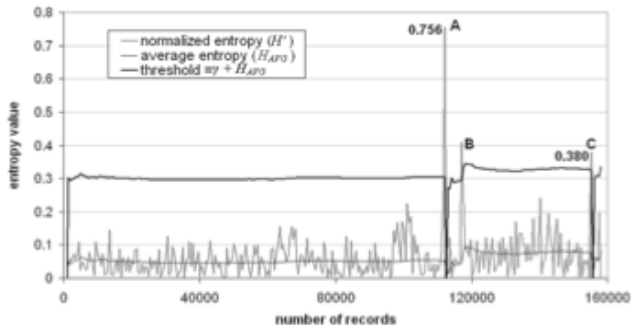


Figure: Entropy values for SpecPhoto.

Classification Accuracy of the Forest - SpecObj

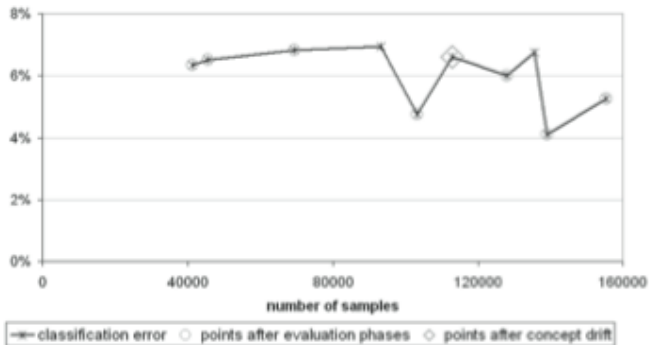


Figure: Forest classification errors for SpecPhoto.

Average Total Times

- Average total times to process synthetic data (32.82 minutes and 32.2 minutes).
- Both recorded times are in the same range.
- Conclusion: Evaluation phases do not add any significant execution time.

Outline

- 1 Motivation
- 2 Background
- 3 Basic Streaming Random Forest Algorithm
- 4 Extending the Algorithm to Handle Concept Drift
- 5 Extending the Algorithm to Handle Random Blocks of Labeled Data
- 6 Implementation and Experimental Settings
- 7 Results and Discussion
- 8 Conclusions

Summary

- They proposed a stream classification ensemble algorithm that:
 - handles concept changes using an entropy-based concept drift detection technique.
 - records new expected classification accuracy after changes are presented in the stream.
 - dynamically adjusts its parameters.

Conclusions




- Key feature:
 - It can decide if the forest under construction is robust enough for deployment when a block of labeled records is not long enough to completely update the current forest.
- Algorithm is fast and incremental.
- Assuming the trees are balanced, it completes a build/update phase with a time complexity of $\mathcal{O}(U \times tree_{min})$.
- Experimental results demonstrated that the algorithm is able to make appropriate decisions each time a block of labeled records is not long enough.



My Opinion

- It is a good application of RF to streaming data.
- There are no direct comparison to other results, they remark their accuracy is better but using different algorithms and different type of data than references.
- They use real data but simulated streaming.
- I don't work with streaming data, so, proposed algorithm is not usefull for my research line.

My Opinion

- Techniques used are interesting:
 - two window paradigm to detect concept drift.
 - margin function - threshold to decide if algorithm is robust enough.
- Limitations:
 - There is not a real application experiment.
 - Some parameters are user defined: number of total trees, n_{min} , δ .
 - Depending on the applications could be difficult to specify.

-  [11] H. Wang, W. Fan, S. Philip, and J. Han, "Mining Concept-Drifting Data Streams Using Ensemble Classifiers," Proc. Ninth ACM SIGKDD, pp. 226-235, Aug. 2003.
-  [12] M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham, "A Multi-Partition Multi-Chunk Ensemble Technique to Classify Concept-Drifting Data Streams," Proc. Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD '09), Apr. 2009.
-  [13] F. Chu and C. Zaniolo, "Fast and Light Boosting for Adaptive Mining of Data Streams," Proc. Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD), pp. 282-292, 2004.

-  [14] W. Street and Y. Kim, "A Streaming Ensemble Algorithm (SEA) for Large-Scale Classification," Proc. Seventh ACM SIGKDD, pp. 377-382, Aug. 2001.
-  [15] Y. Sun, G. Mao, X. Liu, and C. Liu, "Mining Concept Drifts from Data Streams Based on Multi-Classifiers," Proc. 21st Int'l Conf. Advanced Information Networking and Applications Workshops (AINAW), pp. 257-263, May 2007.