On The Effectivensess of the Hoeffding Tree Classifier on Data Streams

Anadi, C. O. & Sennaike O. A.
Department of Computer Sciences
University of Lagos
Akoka, Lagos State, Nigeria
E-mail: chinenye.anadi@yahoo.com, osennaike@unilag.edu.ng
08082468751, 08033322378

ABSTRACT
Improvement in computing technology has enhanced our capability to collect, generate and manage significantly large amounts of data. This explosive data growth has created an urgent need for novel techniques and automated tools that can intelligently transform these huge amounts of data into useful information or knowledge that would aid decision making. Data is being generated at a speed which existing technologies cannot handle, therefore more powerful techniques and tools are being developed to handle these streams of data. One of such technique is the Very Fast Decision Tree (VFDT) or Hoeffding Tree.
In this paper, experiments were carried out using on Hoeffding Tree algorithm to determine its optimal parameters with respect to prediction accuracy. The results show that most of the parameters have little or no effect on the accuracy of the classifier. On the other hand, the Information gain proved to be a better split criterion than Gini index. Also the Naïve Bayes Threshold has predictable effects on the accuracy with larger thresholds producing lower accuracy.

Keywords: Data Stream Mining, Very Fast Decision Tree, Hoeffding Tree, Machine Learning.

1. BACKGROUND TO THE STUDY

Advancements in technology brought about the explosive growth of data. With so much data everywhere, there arose the need to manage and process these data to obtain meaningful information, gain knowledge, and discover previously undetected patterns; that would aid decision making. The need to gain insight from these fast growing data was the motivation in the field of data mining. According to Paidi (2012), data mining is the process of extracting new, meaningful and useful information or knowledge from large data stores. Data mining can also be defined as the analysis of (usually large) observational data sets to discover unsuspected or unknown relationships and to summarize the data in unique ways that is both understandable and useful to the owner. The summaries and relationships discovered through data mining exercises are often referred to as patterns or models; examples include linear equations, rules, tree structures, clusters, and recurrent patterns in time series (Hand, et al., 2001).

Many approaches such as statistics and machine learning are presently being used in this fast emerging field; and with the increase of unstructured databases and the rapid growth of the World Wide Web, new technologies, techniques and applications are continuously being developed (Paidi, 2012). The data mining approach does not solve the problem of a continuous supply of data, which is experienced presently in the large amounts of data continuously generated daily. The speed at which the data is generated, the different types and the amount of data generated has increased tremendously such that even if the data could be stored, it may be difficult or impossible to process the data multiple times due to the increased volume of the incoming data (Aggarwal, 2007).

Adaptable and powerful tools are therefore needed to reveal valuable information (knowledge) from the vast amounts of data. In some cases, the pattern or structure of the data may change over time, so it is necessary to develop the stream mining algorithms that tackle the changing data pattern. This is even more challenging from a computational and algorithmic point of view.

A data stream is a sequence of data items that arrives in timely order. It is different from data in a traditional database, because data streams are a continuous, unbounded flow of data that usually come with high speed and have a distribution that may change with time (Mala & Dhanaseelan, 2011). Data stream mining is extracting valuable information from rapidly continuous data records. Unlike with data mining where the records are in a database, and can be retrieved at any time, data streams are so large and rapidly generated that the available computing resources are insufficient in storing and analyzing the data. So tools for mining these data streams have to be powerful and flexible, as they may only get one or a few passes over the data. Examples of sources of stream data include internet and network traffic, sensor data etc.

The challenges in processing data streams can be categorised into 5 (Kholghi & Keyvanpour, 2011): irregular data arrival rate and variations in arrival rate over time, quality of mining results, limited memory size and huge amount of data streams, limited resources and problems in data analysis. Challenges that have risen in data stream mining have been solved using deep-rooted statistical and computational methods. These solutions can be categorize into data-based and task-based. Data-based solutions involve examining a subset of the whole dataset or transforming the data vertically or horizontally to a smaller-sized data representation. On the other hand, task-based solutions, implement techniques from computational theory to attain time and space efficient solutions (Gaber, et al., 2005).

Three major tasks in data stream mining are data stream clustering, frequent pattern mining and data stream classification (Bifet, et al., 2011). Clustering involved grouping similar records together to form a cluster. The principle of clustering is to maximize class member similarity and minimize interclass similarity. That is, clusters are formed such that objects within a cluster have very high similarity with one another, but are rather dissimilar or have few similarities with objects in other clusters. It is difficult to use regular (data mining) clustering algorithms for data streams because of the one-pass constraints on the data stream. This is because of the continuous arrival of data points, and the need for real time analysis on the data. These features call for incremental clustering and maintaining cluster structures that change over time. The data stream may also continually evolve, and new clusters may form; others may disappear in reaction to the dynamics of the stream (Gama, 2010). Several stream clustering (algorithms) exist (Bifet, et al., 2011). They include StreamKM++, (Ackermann, et al., 2012), CluStream (Aggarwal, et al., 2003), ClusTree (Baldauf, et al., 2009), Den-Stream (Cao, et al., 2006), D-Stream (Tu & Chen, 2008) and CobWeb (Fisher, 1987).

Frequent pattern mining entails finding relationships among the items in a database. The streaming data Frequent Pattern Mining problem tries to find the set of all frequent itemsets (or patterns) within data window. Compared with other stream mining tasks, frequent pattern mining presents three computational challenges. Firstly, there is usually an exponential number of patterns to be considered. For instance, when seeking subsequences, a sequence of length N contains 2N possible subsequences; and if the data streams in quickly, the computational complexity has to be almost linear so as to keep up with the stream (Aggarwal & Han, 2014). Secondly, the memory requirement for frequent pattern mining can be very large. Therefor the stream mining algorithm has to be very memory efficient. Lastly, the algorithm should be able to balance accuracy and efficiency.

Classification is a type of supervised learning where a set of dependent variables are to be predicted based on another set of input attributes. Classification algorithms can usually be divided into training or model building phase and the testing or model testing phase. The most common methods used in data stream classification are probabilistic methods, decision trees, SVM methods, rule-based methods, instance-based methods, and neural networks, Naïve Bayes Classifier and Very Fast Decision Tree (VFDT) (Domingos & Hulten, 2000).

A Hoeffding tree is an incremental anytime decision tree algorithm that is can learn from massive data streams, assuming that the distribution generating samples does not change over time (no concept drift). Hoeffding trees exploit the fact that processing a small sample is often enough to choose an optimal splitting attribute. This idea is supported statistically by the Hoeffding bound (Domingos & Hulten, 2000).

In order to evaluate the algorithm, we use the prediction accuracy. Accuracy is measured as the percentage of correct predictions that a model makes on a given set of data. Hence, the most accurate learning algorithm is the one that makes the fewest mistakes when predicting the class labels of test samples. In order to calculate accuracy, the test dataset must also contain class labels to act as a cross reference for the predictions of the algorithm.

1.1 Statement of Problem

With advancements in both hardware and software technologies, automated data generation and storage has grown, and continues to grow exponentially. For a number of systems, decisions must be made as these data are generated thus the need to ensure that models are tuned with optimal parameters.

1.2. Objective

The purpose of this research is to identify the parameter(s) that has the most influence on the efficiency of the Hoeffding tree algorithm and determine the optimal set of.

2. METHODOLOGY

2.1 Data Collection

The data used for this research is gotten from the University of California, Irvine, machine learning repository (UCI repository). The UCI repository is a collection of databases, domain theories and data generators that is used for empirical analysis of machine learning algorithm by the machine learning community. It has overtime been widely used and cited by researchers, educators and students the world over as a primary source of machine learning data sets.

2.1.1 Data Description

The forest cover type dataset from the UCI KDD Archive was used for the experiments. The data is the forest cover type for 30 x 30 meter cells obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. Independent variables were derived from data originally obtained from US Geological Survey (USGS) and USFS data. The data is in raw form (not scaled) and contains binary (0 or 1) columns of data for qualitative independent variables (wilderness areas and soil types). The data set contains 54 attributes, but 12 measures (10 quantitative variables, 4 binary wilderness areas and 40 binary soil type variables), and a last attribute that is the class label. The following are the variable name, variable type, the measurement unit and a brief description.

Table 1: Variable Information of Cover Type data (The UCI KDD Archive, Information and Computer Science, University of California, Irvine, 1999).

Name	Data Type	Measurement	Description
Elevation	Quantitative	meters	Elevation in meters
Aspect	Quantitative (numeric)	azimuth	Aspect in degrees azimuth
Slope	Quantitative	degrees	Slope in degrees
Horizontal Distance To Hydrology	Quantitative	meters	Horz Dist to nearest surface water features
Vertical Distance To Hydrology	quantitative	meters	Vert Dist to nearest surface water features
Horizontal Distance To Roadways	quantitative	meters	Horz Dist to nearest roadway
Hillshade_9am	quantitative	0 to 255 index	Hillshade index at 9am, summer solstice
Hillshade_Noon	quantitative	0 to 255 index	Hillshade index at noon, summer solstice
Hillshade_3pm	quantitative	0 to 255 index	Hillshade index at 3pm, summer solstice
Horizontal Distance To Fire Points	quantitative	meters	Horz Dist to nearest wildfire ignition points
Wilderness_Area (4 binary columns)	Qualitative (nominal)	0 (absence) or 1 (presence)	Wilderness area designation
Soil_Type (40 binary columns)	qualitative	0 (absence) or 1 (presence)	Soil Type designation
Cover_Type (7 types)	qualitative	1 to 7	Forest Cover Type designation

Wilderness Areas:
1 -- Rawah Wilderness Area
2 -- Neota Wilderness Area
3 -- Comanche Peak Wilderness Area
4 -- Cache la Poudre Wilderness Area

Soil Types:
1 to 40: based on the USFS Ecological Land type Units for this study area.

Forest Cover Types:
1 -- Spruce/Fir
2 -- Lodgepole Pine
3 -- Ponderosa Pine
4 -- Cottonwood/Willow
5 -- Aspen
6 -- Douglas-fir
7 -- Krummholz

2.2 Data Preprocessing

The dataset gotten from the UCI repository was extracted from archived folder (.gz or .zip). The extracted file was saved as a Comma Separated Value (.csv) file. The attribute names were added to the top of the .csv file. Usually, the datasets do not contain the column headers (attribute names); rather, the names of the attributes are in the dataset description in the repository. The .csv file is then converted to .arff (Attribute-Relation File Format) file. An Attribute-Relation File Format (arff) file is an ASCII text file that contains a list of instances sharing a set of attributes. ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software (Paynter & Kirkby, 2008). This preprocessing was carried out on both the training dataset and the test dataset.

2.3 Training or Model Building

The classification algorithm is trained with the training dataset and a model is built. This model would then be used to predict the class labels of the test data set. The validity (quality) of the model depends on the number of data samples used in building it. The more the training samples, the more the chances of building a good model. For this work, the ratios of training data to test data that would be used is 70:30.

2.3.1 Parameters

The Hoeffding Tree algorithm has the following parameters that determine the output. They can be altered to vary the result of the classification.

- **Maximum Byte Size**: This specifies the maximum amount of memory (in bytes) to be used by the decision tree. The sizes used in this experiment are: 67,108,864 bytes (64MB), 33,554,432 bytes (32MB), 16,777,216 bytes (16MB), 8,388,608 bytes (8MB) etc.

- **Numeric Estimator**: The numeric estimators used are Gaussian approximation evaluating, Greenwald-Khanna, quantile summary, VFML method and Exhaustive binary tree

- **Memory Estimate Period**: This specifies the number of data instances between memory consumption checks. The number used in this experiment are 200,000 instances, 500,000 instances, 900,000 instances etc.

- **Grace Period**: This stipulates the number of instances a leaf should observe between split attempts, since it is computationally costly to evaluate the information gain of attributes (split criterion) after every training sample. Also, since a single sample will have little effect on the results of the calculation, it is sensible to wait for more samples before re-evaluating.

- **Split Criterion**: Specifies the criterion for node splitting. The criterion used are Information Gain and Gini index.

- **Split Confidence**: This is the allowable error in split decision, values closer to 0 will take longer to decide. The values used are 0, 0.25, 0.5, 0.75 and 1.

- **Tie Threshold**: This is the threshold below which a split will be forced to break ties. The values used are 0, 0.25, 0.5, 0.75 and 1.

- **Binary Split**: This indicates whether or not binary splits are allowed.
- **Stop Memory Management**: This indicates whether or not stop growing the tree as soon as memory limit is reached.
- **Remove Poor Attributes**: This indicates whether or not poor attributes are to be disabled.
- **No Pre-pruning**: This indicates whether or not pre-pruning is to be disabled.
- **Leaf Prediction**: This specifies the predictor to be used at the leaves. The leaf predictors used are: Majority Class (MC), Naïve Bayes (NB) and Naïve Bayes Adaptive (NBAdaptive)
- **Naïve Bayes Threshold**: Specifies the number of instances that should observed at a leaf before Naive Bayes is permitted. The values used are 0, 40, 80, 120 and so on.

The default parameters of the Hoefffing tree is as shown in the table 2

Table 2: The default value of parameters for the Hoeffding Tree

MaxByteSize	NumEstimat	MemEstPeriod	GracePeriod	SplitCriterion	SplitConfidence
33554432	Gaussian approximation	1000000	200	Info Gain	0

TieThreshold	BinarySplit	StopMemMgt	RemovePoorAtts	NoPrePrune	LeafPrediction	NBThreshold
0.05	no	no	no	No	NB Adaptive	0

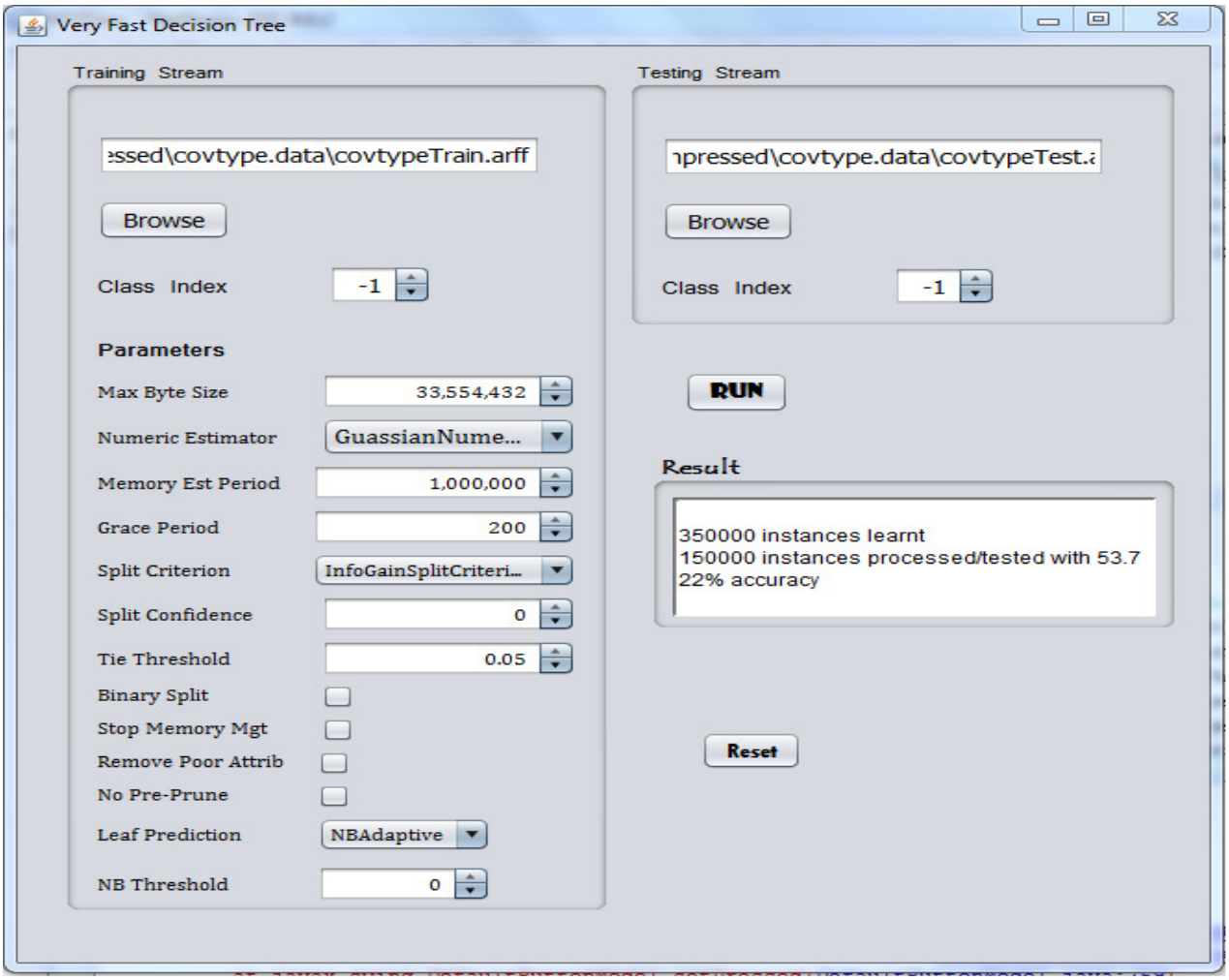

Fig 1: User interface with classification result of default parameters.

3.4 Testing and Model Evaluation

The validity of the model is tested on the test data that is has not been seen by the algorithm. The model predicts the class labels of the records in the test data set, and the accuracy of the prediction is calculated.

3. RESULTS

The default value of the parameters gave an accuracy of 53.722%. Altering the maximum byte size produced no change in the classification accuracy, so did adjusting the Memory Estimate Period, checking the boxes for the Binary Split, Stop Memory Management, Remove Poor Attributes and No Pre-Pruning. The highest accuracy was gotten when the Numeric Estimator was changed to VFML, followed by changing the Leaf Prediction to Majority Class (MC).

The numeric Estimator that produced the highest accuracy was the VFML (66.897%), followed by the Gaussian approximation (53.722%) and then the Greenwald-Khanna quantile (48.747%). For the Split Criterion, Information Gain gave a higher accuracy (53.722%) than Gini Index's 49.396%. For the Leaf Prediction, Majority Class produced 65.576%, Naïve Bayes 53.651% and Naïve Bayes Adaptive 53.722%. Below is a summary of other results obtained.

Table 3: Values of Grace Period and the classifier's accuracy

Grace Period	50	100	200	300	400	500	600	700	800	900	1000
Accuracy (%)	61.323	55.383	53.722	57.383	57.241	61.277	61.587	61.834	60.385	58	61.401

Table 4: Values of Split Confidence and the classifier's accuracy

Split Confidence	0	0.1	0.3	0.5	0.65	0.85	1
Accuracy	53.722	53.225	53.639	55.595	55.727	55.521	54.498

Table 5: Values of Tie Threshold and accuracy

Tie Threshold	0	0.2	0.5	0.501	0.55	0.551	0.6	0.75	1
Accuracy	64.439	53.184	55.591	55.642	55.594	55.598	54.498	54.498	54.498

Table 6: Values of NBThreshold and Accuracy

NBThreshold	0	20	50	100	300	500	1000
Accuracy	53.722	53.722	53.722	53.651	53.298	51.335	51.045

4. DISCUSSION OF FINDINGS

From the result, we observe that the effect of the Naïve Bayes Threshold parameter results produces a predictable effect on the classification accuracy of the Hoeffding Tree algorithm. As the threshold becomes larger, the classification accuracy drops. The other parameters seem unstable. They seem to raise and lower the accuracy of the classifier randomly. For a more accurate classifier, the Information gain is recommended as the split criterion, as it gives a better classification accuracy than Gini index.

5. CONCLUDING REMARKS

The Hoeffding Tree is a decision tree designed for data streams. It is an efficient data stream classifier and does not require a lot of memory, as it does not have to store the data being processed. The memory it requires is just the one for storing the tree. The size of the tree would depend on the training data used. In this work, we observed that the algorithm can work with 32 MB of memory as well as 16 MB, 8 MB, and even 1 MB of memory without affecting the classifier's accuracy. When the data being used induces a large tree to be built, say larger than the allocated memory, the regular memory checks would point out nodes to be deactivated or reactivated which increases the computational time, but leaves the accuracy unaffected. The information gain is also observed to result in better classification accuracy when compared with the gini index.

REFERENCES

1. Ackermann M. R., Märtens, M., Raupach C.,, Swierkot K., Lammersen, C., and Sohler C., (2012). StreamKM++: A clustering algorithm for data streams. J. Exp. Algorithmics 17, Article 2.4 (May 2012), 1.2 pages. DOI=http://dx.doi.org/10.1145/2133803.2184450
2. Aggarwal, C. C., Han, J., Wang, J. & Yu, P. S., (2003). A Framework for Clustering Evolving Data Streams, Proceedings of the 29th international conference on Very large data bases., Volume 29, pp. 81-92.
3. Aggarwal, C., (2007). Data streams: models and algorithms, Springer Science & Business Media, New York.
4. Aggarwal, C. C. & Han, J., (2014). Frequent Pattern Mining, Springer International Publishing.
5. Baldauf, C., Assent, I., Kranen, P. & Seidl, T., (2009). Self-Adaptive Anytime Stream Clustering. Ninth IEEE International Conference on Data Mining, pp. 249-258.
6. Bifet, A., Holmes, G., Kirkby, R. & Pfahringer, B., (2011). Data Stream Mining: A Practical Approach. 2nd ed.,The University of Waikato.
7. Cao, F., Ester, M., Qian, W. & Zhou, A., (2006). Density-Based Clustering over an Evolving Data Stream with Noise, SIAM Conference on Data Mining, Volume VI, pp. 326-337.
8. Domingos, P. & Hulten, G., (2000). Mining High-Speed Data Streams, In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 71-80.
9. Fisher, D. H., (1987). Knowledge Acquisition Via Incremental Conceptual Clustering. Machine learning, II(2), pp. 139-172.
10. Gaber, M. M., Krishnaswamy, S. & Zaslavsky, A., (2005). On-board Mining of Data Streams in Sensor Networks. In: S. Bandyopadhyay, U. Maulik, L. B. Holder & D. J. Cook, eds. Advanced Methods for Knowledge Discovery from Complex Data,Springer London, pp. 307-335.
11. Gama, J., (2010). Knowledge Discovery from Data Streams, CRC Press, Florida.
12. Hand, D., Mannila, H. & Smyth, P., (2001). Principles of Data Mining. Massachusetts, The MIT Press.
13. Kholghi, M. & Keyvanpour, M., (2011). An Analytical Framework for Data Stream Mining Techniques Based on Challenges and Requirements, International Journal of Engineering Science and Technology, III(2), pp. 2507-2513.
14. Mala, A. & Dhanaseelan, F. R., (2011). Data Stream Mining Algorithms: A Review of Issues and Existing Approaches. International Journal on Computer Science and Engineering, III(7), pp. 2726-2732.
15. Paidi, A. N., (2012). Data Mining: Future Trends and Applications. International Journal of Modern Engineering Research, II(6), pp. 4657-4663.
16. Paynter, G. & Kirkby, R., 2008. Attribute-Relation File Format (ARFF). [Online] Available at: http://www.cs.waikato.ac.nz/ml/weka/arff.html [Accessed 15 October 2016].
17. Tu, L. & Chen, Y., (2008). Stream Data Clustering Based on Grid Density and Attraction. ACM Transactions on Computational Logic, I(1), pp. 1-26.