# *Towards a parallel computationally efficient approach to scaling up data stream classification*

# Towards a Parallel Computationally Efficient Approach to Scaling up Data Stream Classification

Mark Tennant, Frederic Stahl, Giuseppe Di Fatta, João Bártolo Gomes

**Abstract** Advances in hardware technologies allow to capture and process data in real-time and the resulting high throughput data streams require novel data mining approaches. The research area of Data Stream Mining (DSM) is developing data mining algorithms that allow us to analyse these continuous streams of data in real-time. The creation and real-time adaption of classification models from data streams is one of the most challenging DSM tasks. Current classifiers for streaming data address this problem by using incremental learning algorithms. However, even so these algorithms are fast, they are challenged by high velocity data streams, where data instances are incoming at a fast rate. This is problematic if the applications desire that there is no or only a very little delay between changes in the patterns of the stream and absorption of these patterns by the classifier. Problems of scalability to Big Data of traditional data mining algorithms for static (non streaming) datasets have been addressed through the development of parallel classifiers. However, there is very little work on the parallelisation of data stream classification techniques. In this paper we investigate K-Nearest Neighbours (KNN) as the basis for a real-time adaptive and parallel methodology for scalable data stream classification tasks.

## 1 Introduction

The problem of dealing with 'Big Data' for 'classification' is that classical methods of classification are not suitable with respect to the new scale, speed and the vari-

Mark Tennant, Frederic Stahl and Giuseppe Di Fatta
School of Systems Engineering, University of Reading, PO Box 225, Whiteknights, Reading, RG6 6AY, UK, e-mail: m.tennant@pgr.reading.ac.uk, F.T.Stahl@reading.ac.uk, g.DiFatta@reading.ac.uk

João Bártolo Gomes
Institute for Infocomm Research (I2R), A*STAR, Singapore, 1 Fusionopolis Way Connexis, Singapore 138632 e-mail: bartologjp@i2r.a-star.edu.sg

ety (possible unstructured format) of 'Big Data'. Traditional data mining methods for classification of static data take several passes through the training data in order to generate the classification model, which is then applied on previously unseen data instances. Streaming models differ from this learning procedure of Train and Test to a system that continuously needs to be evaluated and updated. As the data is often either too fast to process in depth, or too vast to store, data stream classifiers must be naturally incremental, adaptive and responsive to single exposures to data instances. The continuous task of re-learning and adaptation aims to tackle the problem of concept drift [12] (changes of the patterns encoded in the streams over time). An ideal data stream classifier should incorporate certain features [18]: the classifier must limit its size (memory footprint) as streams are theoretically infinitely long; the time taken to process each instance is short and constant so as not to create a bottleneck; each data instance is only seen once by the classifier; the classification model created incrementally should be equivalent to a 'batch' learner given the same training data; and the classifier must be able to handle concept drift. Data streams come in all forms as technologies merge and become more interconnected. Classic applications are: sensor networks; Internet traffic management and web log analysis [13]; TCP/IP packet monitoring [8]; and intrusion detection [15]. However, capturing, storing and processing these data streams is not feasible, as the data stream is potentially infinite. Systems that could analyse these very fast and unbounded data streams in real-time are of great importance to applications such as the detection of credit card fraud [6, 20] or network intrusion. For many data mining problems parallelisation can be utilised to increase the scalability. It is a way for classifiers to increase the speed of both model creation and usage, notable developments are for example the tree and rule based parallel classifiers [16, 22, 23]. Working with data streams limits the processing time available for classifications (both testing and training), to the small window of time in between the arrival of instances. Parallelisation of data stream mining algorithms offers a potential way to create faster solutions that can process a much larger amount of data instances in this small time window and thus can scale up these algorithms to high velocity data streams. One of the currently fastest streaming decision tree based classifiers VFDT (Very Fast Decision Tree) [11] is simple, incremental and has great performance. Unfortunately they are not inherently scalable and lack the ability to be efficiently parallelised. The problem with distributing complex streaming classifier models (such as decision trees) over a cluster, is that it reduces their ability to adapt to concept drift and creates new problems such as load imbalance and time delays. KNN is typically not suited to data stream mining without adaptation (such as employing KD-Trees,P-Trees, L-Trees, MicroClusters) [26], as they incur a relatively high real-time processing cost, proportional to their training data size. In this paper we propose KNN as a basis for the creation of a parallel data stream classifier. The motivation for using KNN is because KNN is inherently parallelisable, for example [25] developed a parallel KNN using the MapReduce parallel programming paradigm [9]. This has been demonstrated in the past for KNN on static data [17], but not yet on data streams. Versions of KNN for data streams exist, such as [10], but to our knowledge there are no parallel approaches for KNN on data streams.

This paper investigates the feasibility of developing a parallel data stream classifier based on KNN. For this we build a simple serial (non-parallel) KNN classifier for data streams that is able to process and adapt on streaming data in real-time. We then show that this basic real-time KNN classifier is competitive (in terms of classification accuracy) compared with other existing well established data stream classifiers, yet very slow. Next we propose a parallel version of this real-time KNN classifier and show empirically that it achieves the same level of classification accuracy compared with its serial version, but has the potential to process data much faster. This parallel real-time KNN method is not only expected to compete well with the current best classification algorithms in terms of accuracy and adaptivity to concept drifts, but also in terms of its scalability to future increases in 'Big Data' in terms of volume and velocity [14].

The remainder of this paper is organised as follows: Section 2 proposes to use KNN as the basis for a data stream classifier that is parallelisable and provides a comparative analysis with other data stream classifiers. We go on to highlight its weakness (processing time), and propose to counter this with parallisation. Section 3 then introduces a parallel version of the KNN based data stream classifier. Section 4 discusses ongoing work and concluding remarks.

## 2 Adapting KNN for Data Stream Classification

This sections outlines a basic KNN algorithm for data streams and evaluates this real-time KNN in terms of its predictive performance compared with other currently leading data stream classifiers. Motivated by the results a parallel version of real-time KNN is proposed in Section 3.

### *2.1 A Simple Real-Time KNN Based Classifier for Data Streams*

When dealing with data streams of infinite length it is apparent that the size of the training set needs to be managed, simply adding more and more items into the training set over time is not feasible for long-term performance. Too many items in the training set will have a negative effect on the performance of the classifier in terms of CPU-time, memory consumption but also accuracy as old potentially obsolete concepts may be used for prediction. Too few items will negatively impact the classifiers accuracy as well, as new emerging concepts may only be covered partially. Loosely speaking, the classifier needs to be able to adapt to 'concept drifts'.

In this section we give a brief overview on the principle of KNN and discuss the *sliding window* approach as a means to manage the training pool and sampling from the data stream. The classifier maintains a training set of previously classified instances. New instances that require classification are evaluated against each instance in the training set using a distance measure indicating how 'similar' the

new (test) instance is to each training instance. Once all training items have been evaluated against the new instance the training instances are re-ordered by their 'distance' evaluation. The top 'K' training instances (the Nearest Neighbours) are used as individual votes with their classification labels. The classification label with the majority vote is assigned to the new instance.

Each training item must be evaluated against the new instances to see how closely they resemble each other. This resemblance is quantified by the squared Euclidean distance.

$$Squared\,Ecludian\,Distance = \sum_{i=1}^{N}(X_i - \bar{X}_i)^2$$

where N is the number of attributes, X is the training record and $\bar{X}$ is the test record.

Categorical attributes are re-quantified to give an equal distance of 1 unit between any different attribute values. More categorical attribute tailored distance metrics could be used, however, for showing that KNN is competitive with other existing data stream classifiers this simple approach is sufficient. A few 'Windowing' techniques exist to manage the number and relevance of training items in a training set.

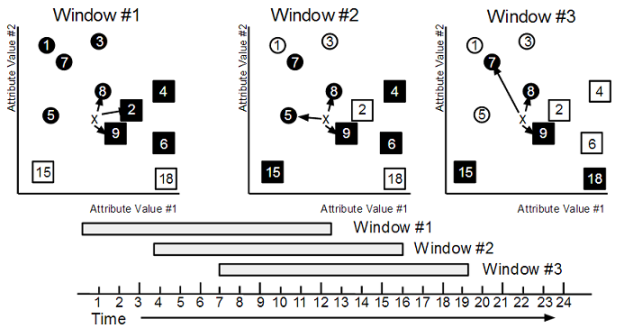


**Fig. 1** Sliding Windows with KNN. The shapes represent the data instances' classes (circle and square) and the number inside the shape is the time stamp. Shaded instances (shapes) are within the current sliding window and available for KNN.

A popular, yet simple technique is the 'Sliding Window' as depicted in Figure 1, where a buffer of predetermined size is created to hold the training instances according to the First In First Out (FIFO) principle w.r.t the timestamps of the instances. Therefore, as a new instance arrives it replaces the oldest instance in the sliding window. As the sliding window size (number of items it contains) is fixed it naturally keeps the training focused upon the recently seen instances without the need of additional management or complex statistical evaluations. Possibly more for KNN tailored Windowing approaches could be developed, however, Section 2's purpose is to show KNN's general suitability for data stream classification rather than opti-

mising windowing. The reader is referred to [13] for a more comprehensive review of windowing techniques and change detectors.

## *2.2 Experimental Analysis of Real-Time KNN*

This Section evaluates the illustrated simple real-time KNN approach (Section 2.1), empirically regarding its potential adaptation as a parallel data stream classifier. Multiple sliding window sizes were used to also evaluate the effects of a larger training set size upon KNN classifiers. As KNN classifiers only utilise training instances as the basis for classification, we would expect that more training instances potentially produce better results. The tradeoff we expect is that the classification times become much larger.

### 2.2.1 Experimental Setup

To evaluate the accuracy and the speed versus the training sizes, multiple sliding window sizes were created (50,100,250,500,1000,2000 and 5000 instances) and evaluated. The KNN classifier was set to a fixed K-value of 3 and utilised a 'Sliding Window' approach. Each simulation was run 6 times. Both, the Mean accuracy of the complete streams and their corresponding Mean processing times were recorded for evaluation. Our KNN Classifier is implemented using the MOA framework [2]. The MOA framework is an open source environment for evaluating data stream classification, clustering and regression algorithms and comparing them against one-another. The framework incorporates the most widely used algorithms for comparison, such as Hoeffding Trees [11] and Naive Bayes classifiers.

In this paper we utilise three of MOA's artificial stream generators with concept drift. Multiple stream generators were created from which new instances are taken from for training and testing. Initially all instances were taken from one data stream generator. As the concept drifts, overlap occurs, increasingly more instances were taken from a second data stream generator, finally, after the concept drift finished, all instances were taken from the second generator. Each stream generator was set to create 35,000 instances sequentially for classification. Three individual tests were performed on each of the streams: *sudden concept drift*, where the concept in the stream switches to a different one immediately after the 10,000$^{th}$ instance, there is no overlap of instances from both generators; *gradual concept drift*, where two concepts coexist over a period of 1,000 instances from the 10,000$^{th}$ instance onwards, instances from both data streams are taken over a period of 1,000 instances; *recurring concept drift*, which are essentially two sudden concept drifts, where the first concept is replaced by the second one from the 10,000$^{th}$ instance onwards and then the second concept is again replaced by the first concept from the 12000$^{th}$ instance onwards. The MOA stream generators were used to generate unique data instances at run time. Each instance generated, conformed to the underlying concept of the

generator at the time of creation. Each instance was tested upon the classifier to log the classifier's performance before being used for training: this is also know as Prequential testing (Test then Train) [4].

The **STAGGER** data stream [21] consists of three attributes, each valuing one of three values. The stream generator classes each instance according to the attributes randomly assigned and the 'function' selected at run-time. Only two possible class labels exist (True and False). Depending on the function preset by the user the generator labels the instances (True or False). The default value of False is used unless combinations of attribute values are met to produce True classifications, dependant on the function selected. For function 1 an attribute pair of 'Red' and 'Small' gives a True value. Function 2 requires an attribute value of 'Green' or 'Circle', while function 3 needs a 'Medium' or 'Large' value. We have arbitrarily chosen function 1 for the initial concept and function 3 for the concept change.

The **SEA** data stream [24] contains three continuous attributes and two class labels. The stream can be set to 1 of 4 function parameters. The function selected determines the value of a sum threshold of the first 2 continuous attribute values (Threshold Values: 8,9,7,9.5). A class label of True is given only if the Threshold level is surpassed, otherwise class label False is given. We have arbitrarily chosen function 1 for the initial concept and function 3 for the concept change.

The **Random Tree Generator** [11] creates a random tree with split points using the attributes associated with the stream. Each tree leaf is labelled with class labels and then all subsequent randomly generated instances are labelled according to their traversal of the tree to a leaf node. In our experiments the random tree(s) comprise five continuous attributes and three distinct class labels. A drift is achieved by simply generating a different random tree.

### 2.2.2 Results and Interpretation

All streams were randomly generated by the MOA artificial stream generators with a finite number of instances (35,000). The classification accuracies are listed in Table 1 for our real-time KNN approach, Naive Bayes and Hoeffding Tree, where the accuracy is the mean accuracy achieved over the whole 35,000 instances.

The categorical synthetic data stream (STAGGER) on all KNN evaluations was observed to perform best with a training set size of 250 (Table 1). This is probably due to the random nature of the generated stream and an overall imbalance of classes in the training instances.

The SEA dataset [24] represents an interesting problem. On first glance the Hoeffding Tree produces a good tree quickly with good classification results. Upon further investigation of the Hoeffding Tree we found that the underlying tree structure is larger than expected. For the simple task of summing two attributes for classification (True if over a set threshold or False if under) the tree structure contained over 40 decision tree branches, many of which were a split decision based upon the 3$^{rd}$ attribute; an attribute that is known to have no relation to the classification decision within the stream generator.

The increase in accuracy of the KNN classifier with respect to continuous attributes is promising. It shows that the real-time KNN approach is competitive with standard popular data stream classifiers (Hoeffding Trees and Naive Bayes), at the cost of processing time. In general the results of SEA and Random Tree suggest that probably even higher accuracies could have been achieved with a larger sliding window (on the expense of processing time).

**Table 1** Simple real-time KNN classification accuracy and overall processing time needed for the 35,000 instances. A sudden (or gradual) concept drift occurs at instance 10,000, the gradual concept drift lasts until instance 11,000. A recurring concept drift appears 12,000 instances. The run-time *s* is listed in seconds.

| Concept | Learner (Training Size) | STAGGER % (s) | SEA % (s) | Random Tree %(s) |
|---|---|---|---|---|
| SUDDEN | Naive Bays | 83.24(0.08) | 94.15(0.11) | 69.87(0.12) |
| | Hoeffding Tree | 96.65(0.20) | 95.59(0.36) | 78.65(0.36) |
| | KNN(50) | 96.53(0.22) | 90.35(0.25) | 68.53(0.21) |
| | KNN(100) | 98.14(0.38) | 92.56(0.37) | 71.86(0.35) |
| | KNN(250) | **99.25(0.90)** | 94.66 (0.94) | 75.03(0.96) |
| | KNN(500) | 98.87(1.76) | 95.70(1.97) | 77.32(2.03) |
| | KNN(1000) | 97.76(3.55) | 96.37(4.33) | 78.75(4.46) |
| | KNN(2000) | 95.58(6.70) | 96.80 (9.05) | **79.84(9.25)** |
| | KNN(5000) | 90.65(15.65) | **97.12(23.53)** | 79.18(24.00) |
| GRADUAL | Naive Bays | 83.24(0.08) | 94.15(0.11) | 69.86(0.11) |
| | Hoeffding Tree | 96.65(0.21) | 95.57(0.33) | 78.36(0.33) |
| | KNN(50) | 96.35(0.18) | 90.33(0.19) | 67.97(0.18) |
| | KNN(100) | 98.14(0.36) | 92.51(0.34) | 71.32(0.35) |
| | KNN(250) | **99.25(0.86)** | 94.63(0.92) | 74.56(0.95) |
| | KNN(500) | 98.87(1.71) | 95.71(1.96) | 77.01(2.00) |
| | KNN(1000) | 97.76(3.48) | 96.35(4.34) | 78.48(4.39) |
| | KNN(2000) | 95.58(6.66) | 96.80(9.02) | **79.75(9.20)** |
| | KNN(5000) | 90.11(15.93) | **97.12(23.42)** | 79.13(23.90) |
| RECURRING | Naive Bays | 83.62(0.08) | 94.20(0.11) | 71.07(0.16) |
| | Hoeffding Tree | 88.00 (0.2) | 95.37(0.27) | 77.60(0.31) |
| | KNN(50) | 95.10(0.18) | 90.31(0.16) | 68.98(0.18) |
| | KNN(100) | 97.17(0.35) | 92.55(0.34) | 72.13(0.34) |
| | KNN(250) | **98.45(0.85)** | 94.64(0.94) | 75.04(0.95) |
| | KNN(500) | 98.01(1.69) | 95.67(1.96) | 77.17(2.03) |
| | KNN(1000) | 96.61(3.40) | 96.37(4.30) | 78.38(4.41) |
| | KNN(2000) | 93.53(6.57) | 96.78(9.08) | **79.14(9.25)** |
| | KNN(5000) | 85.63(15.55) | **97.00(23.50)** | 77.82(24.14) |

As expected, Figures 2 to 4 show that as the sliding window size increases the time taken to process all 35,000 instances increases linearly. This is because test instances have to be compared with a larger number of training instances, which involves more distance calculations. In each graph the STAGGER data stream is quicker to process due to its categorical variables.

We expect a rise in computational cost due to the training size increase. However, KNN is inherently parallel as each training item can be independently evaluated. Motivated by the analysis results of this section, Section 3 discusses how real-time KNN can be parallelised.

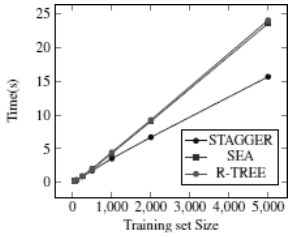


**Fig. 2** Simple real-time KNN processing time with respect to training set size : sudden concept drift

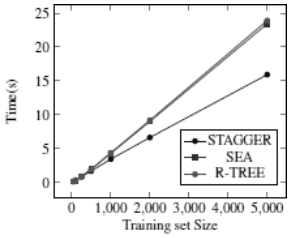


**Fig. 3** Simple real-time KNN processing time with respect to training set Size : gradual concept drift

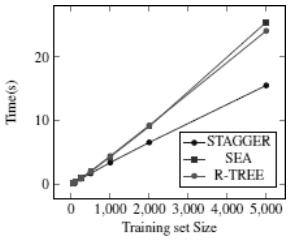


**Fig. 4** Simple real-time KNN processing time with respect to training set size : recurring concept drift

## 3 Parallel Data Stream Classifier Based Upon KNN

As the process of calculating the distance of a test instance to a training instance is an isolated function, with no other information required other than these two instances; it is a data parallel problem. We propose to utilise the open source MapReduce framework (Hadoop) to create a distributed real-time KNN classifier.

### *3.1 The MapReduce Parallelisation Paradigm*

Apache Hadoop [1] is a programming framework and implementation for processing large data, based on Google's MapReduce framework. The MapReduce framework can be simplified into 2 logical steps (Mapping and Reducing). Mapping, refers to the 'bulk' processing that needs to be parallelised. The function to be processed (Map function) is sent to each node (worker) in the cluster, along with a subset of the data to be processed. Each Mapper works independently upon its subset of the data and reports its findings to the Reducer. The Reducer collates all the information from each Mapper and merges it into a logical order (using a Reduce function) before further processing and/or reporting final results. Apache Spark [3]

is an open source implementation of MapReduce tailored for streaming data, which we utilise in this research.

### *3.2 Adaptive Parallel real-time KNN for Data Streams*

To show parallel real-time KNN's classification performance we have created a set of KNN classifiers in the form of Mappers, each employing a sliding window of size:

$$\frac{size\ of\ the\ total\ sliding\ window}{number\ of\ Mappers}$$

Each Mapper holds a unique subset of the training data and an instance of our serial real-time KNN classifier as Map function (described in Section 2). Training upon the cluster involves the additional step of splitting the training data evenly across the cluster. Each Mapper keeps its local training data in its own local 'Sliding Window' to manage its size and recency. As long as additional training data is evenly distributed across the cluster each Mapper will only hold the most relevant (latest) instances. Each Mapper does not have any direct access or communication with other Mappers, or their data. All data input and output is facilitated through the MapReduce paradigm.

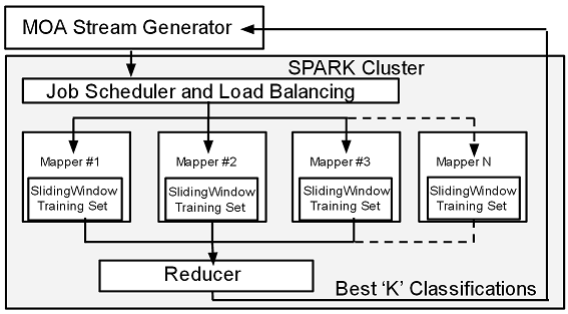


**Fig. 5** Parallel real-time KNN Classifier Architecture.

The complexity of KNN is approximately $O(N)$ where N is the number of training instances. An overview of the cluster layout and process path can be seen in Fig. 5. By distributing the training set out over different Mappers the costly distance evaluations for each of the training items can be calculated in parallel. As shown in Algorithm 1, each Mapper only posts the locally 'K' required best classification / distance metrics pairs. Thus reducing the internal communication of the cluster down to a minimum level, as there is no expensive communication required between the Mappers in order to find the overall 'K' best candidates. This results in $m * K$ instances to be forwarded to the Reducer (if $m$ is the number of Mappers), amongst which the 'K' best candidates are contained. The Reducer phase ensures that the

globally best 'K' instances' class labels are returned to the 'MOA Stream Generator'. This ensures that always the globally 'K' best candidates are returned. Please note that this approach ensures that the parallel real-time KNN will always return the same 'K' best candidates as the serial real-time KNN would, irrespectively of how many Mappers have been used.

**Algorithm 1:** Parallel real-time KNN Process

```
Data: Test Instance
Result: Closest Neighbour(s) Class Labels
Broadcast(Test Instance);
foreach Parallel Mapper KNN Classifier do
    foreach Training Instance in the Sliding Window do
        Evaluate Test Instance( TrainingInstance[i] );
    end
    Sort Evaluations by Distance();
    Send K Best Evaluations to Reducer();
end
Ṙeducer;
foreach MapperResultSet do
    MergeSort(MapperResults);
end
ReturnKBestEvaluations();
```

The scalability of this approach can be broken down into several steps that consume time for both, the maintenance of a sliding window (training data) and performing classifications: communication of instances to the computer cluster $T_{COMIN}$; adding and removing instances to the sliding window $T_{WIN}$; process instances (distance calculations) $T_{DIST}$; communication of the $K$ locally best instances from each Mapper to the Reducer $T_{COMOUT}$; rank the $K \cdot m$ instances and determine the classification based on the majority vote of the $K$ top ranked instances on the Reducer $T_{RANK}$. Thus the total online training and testing time consumed can be summed up as follows:

$T_{TOTAL} = T_{COMIN} + T_{WIN} + T_{DIST} + T_{COMOUT} + T_{RANK}$

whereas we currently parallelised the computationally most expensive steps $T_{WIN}$ and $T_{DIST}$. The times $T_{COMOUT}$ and $T_{RANK}$ are negligibly small, $T_{COMOUT}$ only requires the communication of only $K \cdot m$ instances to the Reducer on a classification request; and $T_{RANK}$ essentially performs a sort on only the $K \cdot m$ instances also only on a classification request. $T_{COMIN}$ is currently not parallelised but required in order to get the training and test data into the cluster. Thus the total time needed is:

$T_{TOTAL} = T_{COMIN} + \frac{T_{WIN}}{m} + \frac{T_{DIST}}{m} + T_{COMOUT} + T_{RANK}$

## *3.3 Initial Experimental Analysis of Parallel KNN Prototype*

### 3.3.1 Experimental Setup

The experimental setup is the same used for the evaluation of the serial real-time KNN in Section 2.2.1, this allows comparing parallel real-time KNN against its se-

rial version. The purpose of this preliminary evaluation is to show that our parallel approach is able to reproduce the same accuracy as serial real-time KNN. The evaluation in this Section is based on a real parallel implementation, however, with only one physical computer node. The empirical scalability analysis is subject to ongoing work.

### 3.3.2 Results and Interpretation

As the results in Table 2 show, our parallel real-time KNN is just as accurate as the serial real-time KNN classifier (see Table 1), but has the ability to be scaled up to faster data streams through the use of a computer cluster. Please note that both, the serial and parallel versions of real-time KNN are deterministic on the same data; and that the parallel version always produces exactly the same classifications as the single real-time KNN classifier does. However, the parallel verison is expected to be faster. The small discrepancies in accuracy between the single real-time KNN classifier and the parallel version exist because MOA's stream generators are non-deterministic.

**Table 2** Parallel real-time KNN Classification Accuracy for 35,000 Instances (Sudden Concept Drift occurs at Instance 10,000. Secondary Concept Drift for Recurring occurred at 12,000 Instances).

| Concept | Learner (Training Size) | STAGGER %(s) | SEA %(s) | Random Tree %(s) |
|---|---|---|---|---|
| SUDDEN | SPARK-NN(50) | 96.87(39.39) | 90.53(41.70) | 67.45(40.86) |
| | SPARK-NN(100) | 98.75(39.03) | 92.65(39.97) | 70.28(41.37) |
| | SPARK-NN(250) | **99.26(40.48)** | 94.37(40.05) | 73.85(42.03) |
| | SPARK-NN(500) | 98.84(40.36) | 95.49(40.06) | 75.70(40.15) |
| | SPARK-NN(1000) | 97.73 (40.12) | 96.22(37.69) | 77.56(38.97) |
| | SPARK-NN(2000) | 95.53(38.98) | 96.34(39.67) | **78.56(40.79)** |
| | SPARK-NN(5000) | 76.02(44.77) | **96.89(39.89)** | 78.46(40.45) |
| GRADUAL | SPARK-NN(50) | 95.88(41.23) | 90.56(41.56) | 66.88(41.35) |
| | SPARK-NN(100) | 97.88(41.97) | 92.64(42.34) | 69.77(41.37) |
| | SPARK-NN(250) | **98.70(41.52)** | 94.30(41.72) | 73.35(42.84) |
| | SPARK-NN(500) | 98.46(41.26) | 95.40(41.87) | 75.23(41.35) |
| | SPARK-NN(1000) | 97.62(41.16) | 96.16(42.25) | 77.23(42.34) |
| | SPARK-NN(2000) | 95.54(41.17) | 96.71(41.82) | **78.40(41.54)** |
| | SPARK-NN(5000) | 90.52(42.80) | **96.83(42.69)** | 78.35(42.86) |
| RECURRING | SPARK-NN(50) | 95.68(40.91) | 90.49(40.43) | 67.76(39.81) |
| | SPARK-NN(100) | 97.76(39.29) | 92.60(40.01) | 70.38(40.06) |
| | SPARK-NN(250) | **98.47(40.57)** | 94.43(39.93) | 73.84(41.33) |
| | SPARK-NN(500) | 97.96(40.17) | 95.47(39.94) | 75.54(41.76) |
| | SPARK-NN(1000) | 96.75(38.87) | 96.19(40.22) | 77.22(39.75) |
| | SPARK-NN(2000) | 93.47(39.90) | 96.67(40.53) | **77.93(40.35)** |
| | SPARK-NN(5000) | 85.58(40.41) | **96.70(41.25)** | 77.09(41.47) |

On average using the same classification metrics and settings on a single machine the SPARK framework adds approximately 40s to the task of classification

regardless of the training sets sizes as can be seen in Table 2. This shows that the times taken for classifications do not follow a linear increase as we would expect. This may be partly due to using MOA and SPARK frameworks together. The MOA framework generates single instances for classification while SPARK works more efficiently with batch broadcasts. The internal communications and data handling of the MapReduce jobs inside the SPARK framework are therefore overly large. As mentioned above, we have used MOA in this implementation in order to evaluate the accuracy of our approach, but it will be removed in the future in order to allow an evaluation regarding parallel real-time KNN's scalability to fast data streams on a real cluster. An interesting feature has been observed with the continuous data sets (Random Tree and SEA). When the training data is set to 1,000 instances, a small reduction in overall time can be seen in the *sudden* and *recurring* results. We believe this is due to the data propagation through the SPARK framework and internal data handling.

Our further works will aim to improve the performance of the classifier. As we have highlighted that our parallel real-time stream classifier performs with similar accuracy as our single real-time classifier we can remove the feedback loop to MAO for testing (see Figure 5) the MOA framework. This should increase the SPARK performance as multiple classification broadcasts and evaluations can be batch transmitted throughout the cluster.

## 4 Conclusions and Ongoing Work

In this research we developed a data stream classifier that is highly parallelisable and hence can be applied to high velocity data streams. So far there are very few approaches of parallel data stream classification algorithms. In our algorithm we have proposed to use KNN as a basis, due to the fact that it is inherently parallel. We have introduced real-time KNN as a simple non-parallel data stream classifier and compared it to existing popular non-parallel data stream classifiers. The results showed that our simple real-time KNN is competitive in terms of the average mean accuracy; however, as expected, on the expense of processing time. Yet, processing time is crucial in many high velocity real-time applications. Motivated by real-time KNN's competitiveness in terms of its mean accuracy we further proposed a parallel version of real-time KNN that is expected to increase the scalability of the classifier to much larger high velocity data streams compared with existing data stream classification algorithms. A prototype implementation of this parallel real-time KNN classifier has been evaluated in terms of its accuracy and it has been found that it achieves a very similar level of accuracy compared with the serial real-time KNN.

We are currently migrating our implementation to a dedicated server cluster. We have created a SPARK cluster of 64 nodes capable of running our real-time parallel KNN classifier. We aim to show that the SPARK overhead can be mitigated with large enough data streams of high velocity. We propose to use both, artificial stream generators and real stream data for parallel classification.

Also a new data structure for real-time parallel KNN, based on 'Micro-Clusters' similar to those used in the CluStream [5] data stream clustering algorithm, is currently in development. The Micro-Clusters are intended to be used in real-time statistical summaries of the training instances in terms of their attributes values and time stamps. This is expected to make real-time KNN as well as its parallel version computationally more efficient in terms of processing times, as well as memory consumption. The second objective of using these Micro-Clusters is to improve the accuracy and speed of adaptation to concept drifts, as Micro-Clusters present a more dynamic way of dealing with the recency of data instances compared with a simple sliding window. Currently the size of the sliding window is set by the user; however, the micro clusters would allow us to take the actual recency of a Micro-Cluster into account as a weight for the distance calculation to test instances. Furthermore Micro-Clusters present a memory efficient way of storing obsolete concepts that could be reactivated quickly in the case an old concept suddenly recurring, which in turn is expected to improve the speed of adaptation in the case of recurring concepts. KNN queries can also be accelerated by using efficient data structures such as multidimensional binary search trees (KD-Trees) [7]. KD-Trees are very efficient and effective for many problems in Computer Graphics and Image Processing where the space dimensionality is intrinsically low. Binary Space Partitioning (BSP) trees are a general technique for recursively dividing a multidimensional space into convex sets by means of hyperplanes. Every non-leaf node of a BSP-tree defines a splitting hyperplane that divides the space into two subspaces. KD-Trees are a particular case where each hyperplane is perpendicular to an axis. BSP-trees can be built by means of approximate Hierarchical Clustering [19] and have been shown to be better suited to improving the efficiency of KNN queries even in high dimensional spaces.

## References


1. Hadoop, http://hadoop.apache.org/ 2014.
2. Massive online analysis(http://moa.cms.waikato.ac.nz), April 2014.
3. Spark: Lightning fast cluster computing (http://spark.apache.org), April 2014.
4. Dawid .A. Stastical theory the prequential approach. *The Royal Stastical Society*, 147:278–292, 1984.
5. C. Aggarwal, J. Han, J. Wang, and P.Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th VLDB Conference*, Berlin Germany, 2003.
6. M. Behdad and Tim French. Online learning classifers in dynamic environments with incomplete feedback. Cancn, Mxico, June 2013. IEEE Congress on Evolutionary Computation.
7. J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
8. T. Bujlow, T. Riaz, and J.M. Pedersen. A method for classification of network traffic based on c5.0 machine learning algorithm. In *Computing, Networking and Communications (ICNC), 2012 International Conference on*, pages 237–241, 2012.
9. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, December 2004.

10. H.D. Dilectin and R.B.V. Mercy. Classification and dynamic class detection of real time data for tsunami warning system. In *Recent Advances in Computing and Software Systems (RACSS), 2012 International Conference on*, pages 124–129, April 2012.
11. P Domingos and G Hulten. Mining high-speed data streams. *KDD*, pages 71–80, 2000.
12. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. A survey of classification methods in data streams. *Data Streams*, pages 39–59, 2007.
13. João Gama. *Knowledge Discovery from Data Streams*. Chapman and Hall / CRC, 2010.
14. John Gantz and David Reinsel. Extracting value from chaos. *IDC iview*, pages 1–12, 2011.
15. A. Jadhav, A. Jadhav, P. Jadhav, and P. Kulkarni. A novel approach for the design of network intrusion detection system(nids). In *Sensor Network Security Technology and Privacy Communication System (SNS PCS), 2013 International Conference on*, pages 22–27, May 2013.
16. M.V. Joshi, G. Karypis, and V. Kumar. Scalparc: a new scalable and efficient parallel classification algorithm for mining large datasets. In *Parallel Processing Symposium*, pages 573–579, Mar 1998.
17. Shenshen Liang, Cheng Wang, Ying Liu, and Liheng Jian. Cuknn: A parallel implementation of k-nearest neighbor on cuda-enabled gpu. In *Information, Computing and Telecommunication, 2009. YC-ICT '09. IEEE Youth Conference on*, pages 415–418, Sept 2009.
18. Domingos. P and Hulten . G. A general framework for mining massive data streams. *Journal of Computational and graphical statistics*, 12, 2003.
19. D. Pettinger and G. Di Fatta. Space partitioning for scalable k-means. In *The Ninth IEEE International Conference on Machine Learning and Applications (ICMLA 2010)*, pages 319–324, Washington DC, USA, Dec 2010.
20. A. Salazar, G. Safont, A. Soriano, and L. Vergara. Automatic credit card fraud detection based on non-linear signal processing. In *Security Technology (ICCST), 2012 IEEE International Carnahan Conference on*, pages 207–212, Oct 2012.
21. J.C. Schlimmer and R.Granger. Beyond incremental processing: Tracking concept drift. *Proceedings of the fifth National Conference on Artificial Intelligence*, 1:502–507, 1986.
22. J. Shafer, R. Agrawal, and M. Mehta. Sprint: A scalable parallel classifier for data mining. *Proceedings of the 22nd VLDB Conference*, 1996.
23. Frederic Stahl and Max Bramer. Computationally efficient induction of classification rules with the PMCRI and J-PMCRI frameworks, 2012.
24. W.N. Street and Y.S. Kim. A streaming ensemble algorithm (sea) for large-scale classi cation. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382, 2001.
25. Chi Zhang, Feifei Li, and Jeffrey Jestes. Efficient parallel knn joins for large data in mapreduce. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, pages 38–49, New York, NY, USA, 2012. ACM.
26. Peng Zhang, B.J. Gao, Xingquan Zhu, and Li Guo. Enabling fast lazy learning for data streams. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 932–941, Dec 2011.