The Incremental Fourier Classifier: Leveraging the Discrete Fourier Transform for Classifying High Speed Data Streams

Chamari I. Kithulgoda[a,*], Russel Pears[a], M. Asif Naeem[a]

[a]*School of Engineering, Computer and Mathematical Sciences*
Auckland University of Technology, Auckland, New Zealand

Abstract

Two major performance bottlenecks with decision tree based classifiers in a data stream environment are the depth of the tree and the update overhead of maintaining leaf node statistics on an instance-wise basis to ensure that classification is consistent with the current state of the data stream. Previous research has shown that classifiers based on Fourier spectra derived from decision trees produce compact array structures that can be searched and maintained much more efficiently than deep tree based structures. However, the key issue of incrementally adapting the spectrum to changes has not been addressed. In this research we present a strategy for incremental maintenance of the Fourier spectrum to changes in concept that take place in data stream environments. Along with the incremental approach we also propose schemes for feature selection and synopsis generation that enable the coefficient array to be refreshed efficiently on a periodic basis. Our empirical evaluation on a number of widely used stream classifiers reveals that the Fourier classifier outperforms them, both in terms of classification accuracy as well as speed of classification.

Keywords: Data Stream, Ensemble Classifier, Discrete Fourier Transform, Concept Drift, Fourier Spectrum, Feature Selection

1. Introduction

The need for scaling up the process of mining high speed data streams is now paramount than ever before. However, greater throughput should not come at the price of prediction accuracy. Incremental learning techniques have been used extensively to address the data stream classification problem and to maintain a good balance between accuracy and efficiency Mena-Torres & Aguilar-Ruiz (2014). In this research we adopt an incremental strategy based on the use of the Discrete Fourier Transform (DFT).

We propose a novel approach for leveraging the DFT to scale up throughput while maintaining or improving classification accuracy over current state-of-the-art data stream classifiers.

The Discrete Fourier Transform has long been a key tool in signal processing and has also been applied to data mining Park (2001), Kargupta & Park (2004), Kargupta et al. (2006). It has several attractive properties for capturing patterns that sets it apart from conventional mechanisms such as decision trees and other types of classifiers. Firstly, it has been shown rigorously that spectra generated from hierarchical classifiers such as decision trees can be represented in compact form thus speeding up the classification process Sripirakas & Pears (2014). Secondly, Fourier spectra have the ability to embed several different patterns (concepts) into one entity unlike conventional ensemble classifier systems which maintain multiple models. This is due to the fact that Spectra can be represented in array form and hence spectra generated at several different points in time can be aggregated into one unifying spectrum that embeds the properties of its constituent spectra. Thirdly, classification can be performed in Fourier space and Fourier spectra, once generated from conventional classifiers, can be used independently of them. Fourthly, the distributive nature of the inverse Fourier transform operation offers the possibility of exploiting parallelism in the classification process.

Such properties have been exploited Sakthithasan et al. (2015), Kithulgoda & Pears (2016) in mining data streams but their use comes at a price. The application of the DFT on multivariate data to produce a spectrum is a non-trivial operation and has time complexity $O(|X|^2)$, where $|X|$ is the size of the feature space Kargupta & Park (2004). The size of the feature space $|X|$ grows exponentially with the dimensionality of the data. In a highly dynamic data stream environment the time spent on repeated application of the DFT at each concept detection point can quickly become prohibitive as our experimentation in Section 6.9 shows.

A much more effective strategy would be to incrementally maintain a spectrum in a fashion analogous to the incremental maintenance of a conventional classifier such

*Corresponding author
Email addresses: ckithulg@aut.ac.nz (Chamari I. Kithulgoda), rpears@aut.ac.nz (Russel Pears), mnaeem@aut.ac.nz (M. Asif Naeem)

as a decision tree. We draw on the Staged Online Learning (SOL) approach proposed by Kithulgoda & Pears (2016) that uses a two staged approach to data stream classification. The SOL approach divides data streams into two types of segments based on the level of volatility in the stream, which is measured by the rate of appearance of new concepts in the stream. Stage 1 represents a high volatility phase while Stage 2 operates in a low volatility phase. We deploy the IFC in the low volatility phase for two reasons. Firstly, refining an already established spectrum by making small incremental changes to it is far more attractive than having to generate a new spectrum from a decision tree. Our experimental results in Section 6.9 supports this premise very strongly. Secondly, the underlying philosophy of the staged approach is that Stage 1 provides a basis for Stage 2 by capturing new patterns as they arrive in a high volatility stage. These patterns, stored as spectra can then be refined in an incremental manner in the low volatility Stage 2 phase without compromising on accuracy.

This research presents an approach that monitors and performs updates at variable-sized windows determined by the rate of change of concepts in the stream. The contributions we make are:

1. an incremental approach for maintaining a spectrum that eliminates the need for regeneration of spectra at update intervals
2. a self indexing hashing scheme that provides fast access to a reservoir that contains a synopsis of changes that occur in a given time window
3. the development of a novel schema pruning scheme which directly targets noisy features in the raw data

The paper is organized as follows. In Section 2 we present background on the properties of Fourier spectra and their derivation from Decision Trees. In Section 3 we give a brief overview of the application of the DFT to multivariate data and give some important insights into the use of the Fourier spectrum as a classifier. In Section 4 we present an incremental approach to maintaining a Fourier spectrum. Section 5 presents an efficient scheme for capturing a synopsis of the data that is critical to the update of the Fourier coefficient array. Section 6 presents our empirical study and in Section 7 we discuss an application of DFT in data engineering context. We conclude the paper in Section 8 with some directions for future work in mining high speed data streams.

2. Background and Related Work

The use of the DFT in data mining has been of recent origin and has been focused on deriving a Fourier spectrum from Decision trees. We first present a basic overview of the derivation of the multivariate DFT from a decision tree and then go on to describe the setting in which our incremental scheme is applied. Before we present the mathematical foundations of the DFT we map fundamental Fourier concepts to their meanings in Table 1 in order to communicate their roles in an intuitive fashion.

Table 1: Mapping of Fourier concepts to their intuitive meanings

Symbol	Meaning
x	A schema consists of a vector of feature values drawn from features that comprise the dataset. A schema is a compact way of defining a set of data instances, all of which share the same set of feature values.
X	The schema set which contains the set of all possible schema for a given dataset.
j	This is a partition of the feature space. Essentially, it is also a vector of feature values, just as with a schema. The only (conceptual) difference is that a schema refers to the data whereas a partition indexes a Fourier spectrum.
J	The partition set that defines the number of coefficients in the spectrum and its size.
$w_{\vec{j}}$	A coefficient in the Fourier spectrum
$\psi_j^{\vec{\lambda}}(\vec{x})$	This is the Fourier basis function that takes as input a feature vector and a partition vector and produces an integer for a dataset with binary valued features or a complex number for a dataset with non-binary feature values.

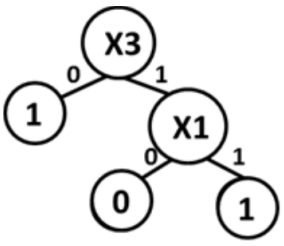
A Fourier spectrum is derived from a Fourier basis set which consists of a set of orthogonal functions that are used to represent a discrete function. Consider the set of all d-dimensional feature vectors where the l^{th} feature can take λ_l different discrete values, $\{0, 1, \ldots, \lambda_l - 1\}$. The Fourier basis set that spans this space consists of $\prod_{l=1}^{d} \lambda_l$ basis functions. Each Fourier basis function is defined as:

$$\psi_j^{\vec{\lambda}}(\vec{x}) = \frac{1}{\sqrt[d]{\prod_{l=1}^{d} \lambda_l}} \prod_{l=1}^{d} \exp\left(\frac{2\pi i x_l j_l}{\lambda_l}\right) \tag{1}$$

where $\vec{j}$ and $\vec{x}$ are vectors of length d; $x(m)$, $j(m)$ are the m^{th} attribute values in $\vec{x}$ and $\vec{j}$, respectively. The vector $\vec{j}$ is called a partition and its order is the number of nonzero feature values it contains.

A function f: $X^d \rightarrow \mathcal{R}$ that maps a d-dimensional discrete domain to a real-valued range can be represented using the Fourier basis functions:

$$f(\vec{x}) = \sum_{j \in X} \overline{\psi_j}^{\vec{\lambda}}(\vec{x}) w_{\vec{j}} \tag{2}$$



x_1	x_2	x_3	f(x)
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

FC	Value
w_{000}	3/4
w_{001}	0
w_{010}	1/4
w_{011}	-1/4
w_{100}	0
w_{101}	0
w_{110}	1/4
w_{111}	0

Figure 1: A Decision tree and its equivalent Fourier spectrum. The data required for classification is given by the table in the middle panel. The IFC classifier only stores and maintains the data given by the table in the right side panel

, where $w_{\vec{j}}$ is the Fourier Coefficient (FC) corresponding to the partition $\vec{j}$ and $\overline{\psi_j}^{\vec{\lambda}}(\vec{x})$ is the complex conjugate of $\psi_{\vec{j}}^{\vec{\lambda}}(\vec{x})$. Henceforth we shall drop the superscript λ from the ψ_j function formulation to simplify the presentation. The Fourier coefficient $w_{\vec{j}}$ can be viewed as the relative contribution of the partition $\vec{j}$ to the function value of $f(x)$ and is computed from:

$$w_{\vec{j}} = \prod_{i=1}^{l} \frac{1}{\lambda_i} \sum_{x \in X} \psi_{\vec{j}}(\vec{x}) f(\vec{x}) \tag{3}$$

In a data mining context, $f(\vec{x})$ represents the classification outcome of a given data instance $\vec{x} \in X$. Each data $\vec{x}$ must conform to a schema and many data instances in the stream may map to the same schema. For example, in Fig. 1, many data instances for schema $(0, 0, 1)$ may occur at different points in the stream. Henceforth in the paper we shall refer to schema instances rather than data instances as our Fourier classifier operates at the schema, rather than at the data instance level. Thus we shall adopt the notation $\vec{x}$ to denote a schema instance, rather than a data instance. The set X is the set of all possible schema, and for the simple example in Fig. 1 it is of size 8.

The absolute value of $w_{\vec{j}}$ can be used as the significance of the corresponding partition $\vec{j}$. If the magnitude of some $w_{\vec{j}}$ is very small compared to other coefficients, we consider the j^{th} partition to be insignificant and neglect its contribution. The order of a Fourier coefficient is simply the order of its corresponding partition. We will use terms like high order or low order coefficients to refer to a set of Fourier coefficients whose orders are relatively large or small, respectively.

The Fourier spectrum of a Decision Tree can be computed using the class outcomes predicted by its leaf nodes. As an example consider the decision tree in Fig. 1 defined on a binary valued domain consisting of 3 features. Its truth table derived from the predictions made by the tree and the corresponding Fourier spectrum that results appears in Fig. 1.

As discussed in the Introduction the Fourier spectrum derived from a decision tree is compact. This is due to the two following properties:

1. The number of non-zero coefficients is polynomial in the number of features represented in the tree Kargupta & Park (2004).
2. The magnitude of the coefficients $w_{\vec{j}}$ decreases exponentially with the order of the partition $\vec{j}$ Kargupta & Park (2004), Kargupta et al. (2006).

These two properties collectively make a spectrum derived from a tree very attractive. Firstly the tree provides a natural filtering mechanism as typically only a fraction of the features have sufficient information gain to be represented in the tree. Once the tree is in place, only the set of low order coefficients defined from partitions appearing in the tree make significant contributions to the classification outcomes.

Kargupta and Park in Kargupta & Park (2004),Kargupta et al. (2006) made use of spectral energy to derive a cut-off point for coefficient order. Given a spectrum s, its energy E is defined by: $E = \sum_{\vec{j} \in s} |w_{\vec{j}}^2|$. For a given energy threshold T, the subset of s (in ascending spectral order) whereby $E \geq T$ is retained; all other coefficients are deemed to be zero and removed from the array. Thus for example in the spectrum defined in Fig. 1 the first order coefficients contain $\frac{9+0+1+1}{9+0+1+1+0+0+1+0} = 91.7\%$ of the total energy and so with a threshold of 90%, only coefficients $w_{000}, w_{001}, w_{010}, w_{100}$ should be retained, thus halving the size of the spectrum that needs to be maintained.

Moreover, as discussed in Kargupta & Park (2004) and implemented in Sakthithasan et al. (2015), spectra can be aggregated with each other. In Sakthithasan et al. (2015) aggregation of spectra was implemented via a pair-wise algebraic summation of the spectra involved as given in Eq. 4:

$$\begin{aligned} s_c(x) &= \sum_i A_i \sum_i s_i(x) \\ &= \sum_i A_i \sum_{j \in P_i} {\omega_j}^{(i)} \overline{\psi}_{j(x)} \end{aligned} \tag{4}$$

where $s_c(x)$ denotes the ensemble spectrum produced from the individual spectra $s_i(x)$ produced at different points i in the stream; A_i is the classification accuracy of its

corresponding spectrum and P_i is the set of partitions for non zero coefficients in spectrum s_i.

Aggregation of spectra brings with it two major benefits. Firstly, a reduction in space as coefficients common to spectra being aggregated need to be stored only once. Secondly, as demonstrated in our empirical study, aggregation performs a similar role to an ensemble of models and leads to better generalizability to new data arriving in the stream.

Once a Fourier spectrum is derived from a decision tree it can fully replace the latter as classification of a newly arriving instance $\vec{x}$ can be computed by applying the inverse transform given in Eq. 2 over the set J that contains the reduced set of coefficients that survive the energy thresholding process.

3. Advantages of the Fourier classifier over the decision tree classifier

We have chosen to contrast our proposed classifier with the decision tree classifier. The decision tree classifier has been shown in a number of empirical studies to be one popular choice for a data stream environment Gama et al. (2003), Kargupta & Park (2004) and Bifet et al. (2009). This is due to two important reasons. Firstly, decision tree induction is efficient, and when used in conjunction with the Hoeffding bound can adapt to new patterns in a data stream with a single pass through the data. Secondly, they capture dependencies between features without the need for time consuming graph traversal, such as in neural networks or Bayesian networks.

3.1. Decision tree overhead

Although decision trees are efficient when compared with other types of conventional classifiers they still have shortcomings that negatively impact on performance. The depth of the tree directly impacts classification performance as the label for a data instance can only be determined by a traversal from the root node to the node that matches with the schema of the given data instance. The deeper the tree, the greater is the number of intermediate nodes that need to be traversed to extract the required class label.

Furthermore, once the true label for a data instance is known, the sufficient statistics, n_{fvc} Domingos & Hulten (2000) which record the number of data instances for each feature f taking value v that belong to class outcome c, need to be updated. These statistics are required to determine whether a leaf node should split into two or more decision nodes. The average time complexity of this update is:

$$O(dvc) \tag{5}$$

where d is the dimensionality (number of features in the data), v is the average cardinality taken over all features and c is the number of classes. This update needs to be performed on a per instance basis even though splits may only occur in a small percentage of cases, thus exposing a fundamental performance weakness of the decision tree method when operating in a data stream environment. In many implementations of decision tree classifiers for a data stream environment a forest of trees is used and this will further increase overheads by a factor proportional to the size of the forest. Thus overall, the appearance of each data instance involves an overhead of $O(dvc)$ in the case of a decision tree.

3.2. Fourier classification overhead

The Fourier classifier does not rely on a deep hierarchical tree structure but instead uses a self indexing hashing scheme to store its schema values. Access to schemas are provided via a shallow two level extendible hashing scheme. The biggest advantage of such a two level structure is that classification is very efficient when the schema for the data instance to be classified is cached in the repository. However, when the spectrum is updated on a periodic basis, classification needs to be performed once again for data arriving after the update point. In this case classification is performed via the IFT operation.

If s is the size of the coefficient array (number of coefficients) then the time complexity of IFT from Eq. 2 is:

$$O(sd^2) \tag{6}$$

as d^2 multiplication are needed for the computation of the vector product between $\vec{x}$ and $\vec{j}$ for each of the s coefficient in the array.

Although a direct comparison between the decision tree and the Fourier classifier is not possible in terms of time complexity we observe that there is no control over any of the factors governing Eq. 5, whereas in Eq. 6 the time complexity can be controlled by limiting the size s of the array as well as the dimensionality of the data. Pruning of the coefficient array can be accomplished effectively by energy thresholding as proposed by Kargupta & Park (2004) and applied in Sakthithasan et al. (2015). In addition to energy thresholding, in Section 4 of this paper we propose a schema pruning scheme that complements on the energy thresholding scheme proposed by Kargupta & Park (2004).

In this research we make use of two types of pruning. Firstly, by reducing the size (s) of its coefficient array it reduces the time complexity of the IFT, leading to smaller classification times after a spectral update is performed.

The second type of benefit is that it reduces overhead by excluding some of the d features from the computation of the vector product between $\vec{x}$ and $\vec{j}$. This is done by eliminating features that do not contribute to the classification process. The details are described in Section 5.1.

3.3. Fourier coefficient array update overhead

Just as with the decision tree the Fourier classifier needs to update its model by refreshing the coefficient array. Each coefficient of the array plays a role similar to the leaf node of a decision tree classifier but there is no

need to store sufficient statistics for each coefficient apart from its numeric value. Thus the cost of accessing a large 3 dimensional array is avoided during the model update operation.

From Eq. 3 we observe that the time complexity for the update operation is $O(nd^2|X|)$ where n is the number of coefficients affected by the update and X is the schema set. Each coefficient to be updated requires $d^2 \times |X|$ multiplications to compute the new value of the coefficient that is being updated.

Our strategy for reducing the update overhead is two-fold. Firstly, as noted in Section 3.2 above, feature selection enables us to eliminate some of the d features from the vector product operation. The reduction factor R_1 obtained through feature selection is:

$$R_1 = \frac{|X|}{|F|} \tag{7}$$

where $|X| = \prod_{i \in X} \lambda_i$ and $|F| = \prod_{i \in F} \lambda_i$ where F denotes the subset of features (drawn from feature set X) that survive the feature selection process.

Secondly, Theorem 1 in Section 4.1 enables us to prune the schema set X. It does this by identifying a subset C of X consisting of those schema that change their class labels from the last observed interval. The Theorem ensures that the coefficient array can be updated efficiently without compromising on classification accuracy. Theorem 2 further constrains the size of C when certain conditions are met.

If C is the subset of X identified by Theorem 1 (which we present in Section 4) over which the update operation must be performed, then the reduction factor obtained through schema pruning is:

$$R_2 = \frac{|X|}{|C|} \tag{8}$$

In stream segments that exhibit a low level of change we expect that only a small fraction of schema will change their class labels - i.e $|C| \ll |X|$ and so $|R_2| \gg 1$. The reduction factors R_1 and R_2 are independent of each other as the class label change rate for a given schema (from one interval to another) is an inherent property of the data and is not influenced by the information content of individual features. Thus the cost of the update operation reduces by a factor $R_1 \times R_2$ with the help of the two optimisations. Our experimentation in Section 6 will quantify the performance gains from feature selection and the incremental coefficient update strategy.

Despite the optimizations introduced by feature selection in Section 5.1 and the incremental approach in Section 4 to Fourier spectrum maintenance, certain types of environments may lead to high resource overhead. These include high dimensionality data on which feature selection is ineffective and returns a large fraction of the original set of features. Also, in highly dynamic streams, the rate of change of class labels to schema would be high, thus triggering more frequent updates to spectra and reducing the effectiveness of the incremental approach.

4. Incremental Approach to Fourier Spectrum Maintenance

As mentioned in Sections 1 and 3.3 we propose to deploy the Fourier classifier in low volatility segments of a data stream. Before we describe the conditions under which the deployment occurs we first define what we mean by drift rate, volatility and the concept of a highly dynamic data stream.

Definition (Drift Rate) : *drift rate is defined as the average number of concept changes that takes place in a given time interval.*

Definition (Volatility) : *Volatility is defined as the average number of new concepts that appear in the stream in a given time interval.*

Thus drift rate and volatility both measure the concept shift activity in the stream, the difference being that volatility discriminates between new concepts and those that have already appeared in the stream.

Definition (Highly Dynamic Data Stream) : *A highly dynamic data stream is one where a combination of sample arrival rate and concept drift rate is sufficient enough to cause system CPU utilization to rise above a maximum tolerable user defined threshold.*

In this situation the system is transiting to a state where normal stream processing would be disrupted unless some extra processing resources can be used.

With the above definitions in place we are now in a position to describe the Staged Online Learning (SOL) approach proposed by Kithulgoda & Pears (2016). The SOL approach is summarized in Fig. 2.

The SOL approach tracks stream volatility and makes use of a rigorous statistical test to determine when the stream transits to a low volatility state. In the low volatility state the overwhelming percentage of concepts appearing in the stream consists of recurrences of previously seen concepts in their original form, or with small incremental changes to their previous occurrence. In such an environment the most efficient approach would be to refresh the array on a periodic basis. Before presenting our strategy for incremental maintenance we will first discuss the research setting in which our strategy is to be deployed. We envisage that in many real-world settings a data stream will experience volatility shifts on a periodic basis. We adopt the definition of volatility within a stream context given by Kithulgoda & Pears (2016).

In any given stream segment of length l if p new concepts appear then the volatility is defined as the ratio $\frac{p}{l}$.

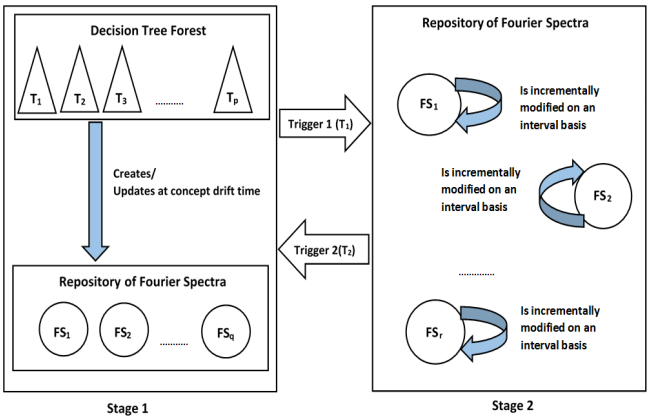


Figure 2: Staged Transition Learning Framework adapted from Kithulgoda & Pears (2016)

The definition is based on the appearance of *new* concepts and not on the rate of concept drift. The rate of concept drift reflects the frequency of switching between concepts. If q switches in concepts take place in a stream segment of size l, then in general only a fraction $p(< q)$ of them will be new and hence the volatility rate defined above as $\frac{p}{l}$ will be less than the rate of concept drift, $\frac{q}{l}$.

In common with Kithulgoda & Pears (2016) we believe that a given stream will cycle between periods of high volatility and low volatility, referred to as Stage 1 and Stage 2 respectively for the rest of this paper. In Stage 1 a conventional classifier such as a decision tree forest can be used to learn concepts as they appear in the stream. Coupling such a learner with a drift detector enable concept boundaries to be determined and when drift is signalled the tree can be transformed into a Fourier spectrum as outlined in Section 2 and then stored in an online repository.

Once the repository is populated with spectra they can be used for classification by applying the inverse Fourier transform as described in Section 2. A significant shift away from relying on decision trees for classification and shifting to Fourier spectra in the repository is signalled by trigger T_1, as shown in Fig. 2. Trigger T_1 tracks the ratio of the number of visits to the ensemble of decision tree to the number of visits to the repository. When classification is dominated by the spectra in the repository the conclusion is that the stream has entered Stage 2 with trigger T_1 firing. It is in such a state that the Fourier Classifier will be deployed.

The trigger T_2 in Fig. 2 is used to detect deviation of spectra in the repository from their initial versions. If the degree of deviation exceeds a certain threshold then this indicates that the stream has re-entered a high volatility state which is signalled by trigger T_2 firing. In the event that Stage 1 is re-activated, a forest of decision trees is grown.

In Stage 2 we envisage that concept drift still takes place but changes in concept occurs for the most part on concepts that have appeared before and not on entirely new concepts. In Kithulgoda & Pears (2016) classification was performed by identifying the spectrum S with the highest classification accuracy for the current concept and then transforming S into a decision tree that learns the changes occurring in a conventional manner. The difference in the learning strategy employed in Stage 2 is that a single tree is used instead of a forest of trees thus saving time in both the learning and classification processes. The experimental results presented in Kithulgoda & Pears (2016) showed a significant reduction in runtime and memory consumption for such an approach.

Even though the staged learning approach yielded good savings in time and memory, several inefficiencies remain. Firstly, in the low volatility state the spectrum (S) that most closely fits is used to induce a decision tree (T). This induction process can involve high computational overhead, depending on the actual size of the spectrum.

Secondly, when concept drift takes place from an existing concept C to a new concept C', the tree T that has evolved to learn changes in the former concept C will need to be converted to spectral form so that it can be stored in the repository for future use if C recurs. At the same time a conversion from S' to tree T' is needed in order to learn future changes in the new concept C'. Such conversions from spectrum to tree and tree to spectrum can be avoided by learning changes in the raw data and translating such changes to the spectrum without the need for transformations to/from the tree domain.

There has been very little research done on optimizing the learning process within the Fourier domain Kargupta & Park (2004). As Park (2001) says learning directly in the Fourier domain is challenging in a high dimensional data environment and optimization strategies such as gradient descent were used to update the coefficient array. In this research we show that there is no need for using such high computational complexity methods.

4.1. Incremental Maintenance of Spectra

Spectra in the repository can be maintained in one of two ways. The first approach taken by Sripirakas & Pears (2014), Sakthithasan et al. (2015), Kithulgoda & Pears (2016) is to select the best performing tree from a decision tree forest at each concept drift point and apply the DFT to produce a spectrum. This spectrum is then aggregated with the most similar spectrum already resident in the repository through the use of a Euclidean distance measure. There are two major issues with this approach. The first is the high overhead in applying the DFT. Secondly, the maintenance operation is not incremental as an existing spectrum is updated in its entirety with a newly generated spectrum from the best performing tree.

The second approach to maintenance exploits the fact that in any given stream segment only a subset of schema instances will experience a change in their class values over the previous segment. Thus, instead of treating the spectrum that covers the data in the newly arriving data seg-

ment as being a new spectrum in its own right, we treat it as an extension of the spectrum that was generated from the previous segment.

The Fig. 3 illustrates the incremental update process that is carried out periodically in intervals. Schema instances x that have changed their class labels $f^{current}(x)$ in the current interval are used in conjunction with $f^{previous}(x)$ from the previous interval, together with the current version of the spectrum array $w_{\vec{j}}^{current}$ to compute the new version $w_{\vec{j}}^{new}$ that will be used in the next interval.

Theorem 1 qauntifies the update computation and provides a proof of its correctness.

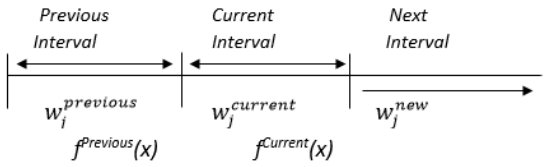


Figure 3: Periodic update of spectra

Theorem 1: Let C be the change set containing all schema instances that have had their class labels changed since the previous update point. The new version of the coefficient array $w_{\vec{j}}^{new}$ can be derived from its current version $w_{\vec{j}}^{current}$ as follows:

$$w_{\vec{j}}^{new} = w_{\vec{j}}^{current} + \frac{1}{|X|}\sum_{x\in C}\psi_{\vec{j}}(x)(f^{current}(x) - f^{previous}(x)) \tag{9}$$

where $f^{current}(x)$, $f^{previous}(x)$ are the class labels in the current and previous intervals respectively for a given schema x.

Proof

$$\begin{aligned}
w_{\vec{j}}^{new} &= \frac{1}{|X|}\sum_{x\in X}\psi_{\vec{j}}(x)f^{current}(x) \\
&= \frac{1}{|X|}\sum_{x\in X}\psi_{\vec{j}}(x)(f^{current}(x) - f^{previous}(x)) \\
&+ \frac{1}{|X|}\sum_{x\in X}\psi_{\vec{j}}(x)f^{previous}(x) \\
&= \frac{1}{|X|}\sum_{x\in C}\psi_{\vec{j}}(x)(f^{current}(x) - f^{previous}(x)) \\
&+ \frac{1}{|X|}\sum_{x\in X\setminus C}\psi_{\vec{j}}(x)(f^{current}(x) - f^{previous}(x)) \\
&+ w_{\vec{j}}^{current}, \text{ as } w_{\vec{j}}^{current} = \frac{1}{|X|}\sum_{x\in X}\psi_{\vec{j}}(x)f^{previous}(x) \\
&= w_{\vec{j}}^{current} + \sum_{x\in C}\psi_{\vec{j}}(x)(f^{current}(x) - f^{previous}(x)) \\
&\text{as } f^{current}(x) = f^{previous}(x)\forall x \in X \setminus C
\end{aligned}$$

In certain cases it is possible to optimize further by restricting the refresh operation to only a subset of coefficients in the coefficient array. Theorem 2 establishes when this optimization applies and how this subset can be identified.

Definition 2 (Pivot) *A feature p is said to be a pivot on the schema set X if $\forall x \in C\ \exists y \in C$ such that*

1. *x and y differ only in index position p*
2. *$f(x) = f(y)$*
3. *$\bigcup_{x(p)\in C} = \{0, 1, \ldots, \lambda_p - 1\}$ where $x(p)$ is the p^{th} position of a schema instance x*

Suppose a pivot feature p exists with change set C. Theorem 2 below identifies the spectrum coefficient subset that is unaffected by the class label changes amongst schema recorded in the change set C.

Theorem 2: If a feature p is a pivot in the previous interval and remains as a pivot in the current interval then the set of coefficients U that is unchanged is given by $w_{\vec{j}}$ with $j(i) = *\ \forall i \neq p$ and $j(i) \in \{1, 2, \ldots, \lambda_p - 1\}$ when $i = p$.

Proof

Consider an arbitrary $\vec{j} \in U$

$$\begin{aligned}
&\psi_{\vec{j}}(x) = \sum_{\vec{x}\in C}\prod_{l=0}^{d-1}\cos\Big(\frac{2\pi j(l)x(l)}{\lambda_l}\Big) \\
&+ i\sum_{\vec{x}\in C}\prod_{l=0}^{d-1}\sin\Big(\frac{2\pi j(l)x(l)}{\lambda_l}\Big) \\
&\text{Define } F = \{0, 1, \ldots, d-1\} \\
&\text{Let } t_1 = \sum_{\vec{x}\in C}\prod_{l\in F}\cos\Big(\frac{2\pi j(l)x(l)}{\lambda_l}\Big) \qquad (10) \\
&\text{and } t_2 = \sum_{\vec{x}\in C}\prod_{l\in F}\sin\Big(\frac{2\pi j(l)x(l)}{\lambda_l}\Big) \\
&\text{Let } m = \prod_{l\in F\setminus\{p\}}\cos\Big(\frac{2\pi j(l)x(l)}{\lambda_l}\Big) \\
&\text{Then } t_1 = \sum_{x(p)|\vec{x}\in C} m\cos\Big(\frac{2\pi j(p)x(p)}{\lambda_p}\Big)
\end{aligned}$$

We are interested in coefficients $j \in U$ such that $j(p) \neq 0$ Now with $k = j(p)x(p)$ and $N = \lambda_p$, Eq. 10 can be written in the form: From Knapp (2009) we have:

$t_1 = m\sum_{k=0}^{N-1}\cos\left(\frac{2\pi k}{N}\right) = 0$ for all integers k and N.

Likewise, $t_2 = m\left(\sin\left(\frac{2\pi j(p)\times 0}{\lambda_p}\right) + \sin\left(\frac{2\pi j(p)\times 1}{\lambda_p}\right)\right.$

$\left. + \ldots + \sin\left(\frac{2\pi(j(p)\times(\lambda_p - 1))}{\lambda_p}\right)\right)$

$= m \times 0$ as $\sum_{k=0}^{N-1}\sin\left(\frac{2\pi k}{N}\right) = 0$

for all integers k and N, also from Knapp (2009).

Thus $\psi_{\vec{j}}(x) = 0\ \forall x \in C$

From Theorem 1 above we have $\forall j \in U$,

$$\begin{aligned} w_{\vec{j}}^{new} &= w_{\vec{j}}^{current} \\ &+ \sum_{x\in C}\psi_{\vec{j}}(x)(f^{current}(x) - f^{previous}(x)) \\ &= w_{\vec{j}}^{current} \text{ as } \psi_{\vec{j}}(x) = 0\ \forall x \in C \text{ and } j \in U \end{aligned}$$

The utility of Theorem 2 is illustrated in the following example which uses binary data to maintain simplicity.

Suppose we have a 3 dimensional dataset with X_2 as a pivot in the previous interval. The Theorem then deals with scenarios such as given below:

Class labels in previous interval: $f(**1) = 0, f(**0) = 1$. In this case, feature X_2 is a pivot in the previous interval as $f(*0*) = f(*1*)$.

Now suppose that the class labels in the current interval are: $f(**0) = 1, f(0*1) = 0, f(1*1) = 1$.

Hence change set $C_{PreviousToCurrent} = \{101, 111\}$.

We observe that feature X_2 remains as a pivot in the current interval.

Now from Theorem 2 we can conclude that coefficients given by $w_{\vec{(*1*)}}$ do not need to be updated.

5. Hashing and Reservoir Management

5.1. Schema Hashing through Feature Selection

Our feature selection strategy is based on identifying the subset of features that play a critical role in the classification process. In the context of classification via a Fourier spectrum, feature selection means the selection of features which contribute to the Inverse Fourier Transform (IFT) operation, as given in Eq. 2. Kargupta and Park in Kargupta et al. (2006) proved that when a feature X_k does not appear in a decision tree then all coefficients $w_{\vec{j}}$=0, whenever $j(k)$ take a non zero value, irrespective of the values taken by other elements of $\vec{j}$. With respect to coefficients with $j(k) \neq 0$ their contribution to the IFT is hence zero.

Further to this, consider the remaining set of Fourier coefficients where $j(k) = 0$ at feature X_i's position. Even though the Fourier coefficients w_j take non zero values we show below that the Fourier basis function has no contribution from such coefficients. On this basis, we conclude that features that do not appear in decision trees in Stage 2 play no part in the classification process and we exploit this property when obtaining the hash function as described below.

From Eq. 1 which we apply on the subset of coefficients where $j(k) = 0$. If $k \neq 1$, $\vec{j}$ can be reordered to make k=1 then without loss of generality.

$$\psi_j^{\vec{\lambda}}(\vec{x}) = \frac{1}{\sqrt[d]{\prod_{l=1}^{d}\lambda_l}}\prod_{l=1}^{d}\exp\left(\frac{2\pi i x(l)j(l)}{\lambda_l}\right)$$

Let $r(l) = \frac{2\pi i}{\lambda_l}$

Now $\psi_j^{\vec{\lambda}}(\vec{x}) = \exp\left(r(1)x(1)\times 0\right)\frac{1}{\sqrt[d]{\prod_{l=2}^{d}\lambda_l}}\exp\left(r(l)x(l)j(l)\right)$

$$= 1 \times \frac{1}{\sqrt[d]{\prod_{l=2}^{d}\lambda_l}}(\exp\left(r(l)x(l)j(l)\right))$$

Thus the Fourier basis function's contribution to coefficients where $j(i) = 0$ has not been affected by the X_i^{th} feature.

We perform feature selection in Stage 1 prior to transformation of a decision tree to its corresponding spectrum. All features that appear in a tree are stored along with its corresponding spectrum. When a new spectrum S_{new} is generated from a winner decision tree at a drift point in Stage 1, a decision is taken whether to store it on its own in the repository or to aggregate it with an existing S already in the repository.

Once feature selection has been performed, a hash scheme for efficiently storing schema instances can be devised. Given a schema instance, a hash function for the i^{th} spectrum S_i is the string $\bigcup_{j=1}^{n} Y(j)$, where $Y(j) = X(j)$ if feature j survives the feature selection process, otherwise $Y(j) = 0$. The rationale for this is that all schema instances will take the same class value regardless of the non contributing feature values. Hence only one instance indexed by a position of 0 for such feature will suffice. For instance, if we consider the binary tree given in Fig. 1, X_1 and X_3 are influential features but not X_2 as it does not appear in the tree and hence does not play a

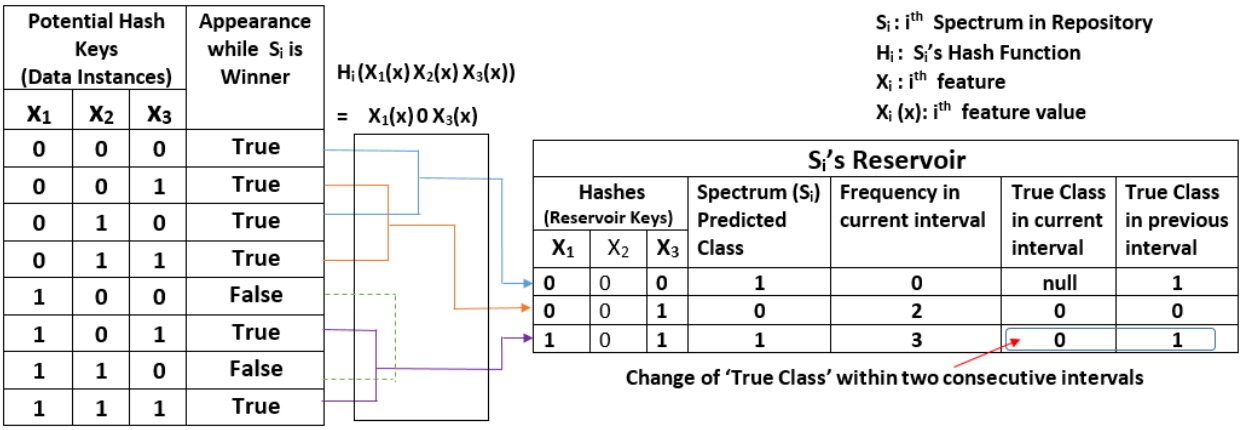


Figure 4: Hashing and Reservoir Structure

part in the classification process. Thus it follows that the hash function for the spectrum is $H(X_1(x)X_2(x)X_3(x)) = (X_1(x)0X_3(x))$.

5.2. Reservoir Organization

A critical element of incrementally adapting the Fourier spectrum to changes in the data stream is a dynamic data structure that captures changes in the schema over a period of time. The Fig. 4 shows a dynamic reservoir structure that keeps track of evolution of schema over time based on an extendible hash implementation. Each and every spectrum in the repository has its own reservoir built using the hashing scheme described earlier.

The reservoir is compact for two reasons. First of all, it only stores schema and not data instances, in a manner analogous to a decision tree that only stores decision paths and not actual data instances. Secondly, feature selection ensures that the worst case space complexity of the reservoir is $O(|X|)$. Thus for the example shown in Fig. 4, the appeared schema reduces in size from 6 to 3 as feature 2 plays no part in classification and hence both 0 and 1 occurrences of this feature maps to the same class, thus requiring only one representation for this feature, which we choose to be 0.

Although the reservoir is compact it can also be sparse as only a subset of schema instances may actually manifest in the stream. For our example schema set, although there are a total of 8 possible schema instances, only 6 may actually appear in the stream, as illustrated in Fig. 4. This calls for a dynamic data structure that only utilizes space as and when needed and hence our use of a hashing scheme. The extendible hashing scheme also ensures that an entry in the reservoir can be extracted with no more 2 memory fetches.

Each spectrum in the repository will be stored its own reservoir according to the structure defined in Fig. 4. At any given point in time, only the winner spectrum undergoes change; all other spectra in the repository remain static until one of them emerges as the winner. Each entry in a reservoir is indexed by its hash value, the class value of the schema at its first appearance, its frequency of appearance, the true class label within the current interval if it appeared, and finally the true class label within its previous appearance, if any.

The reservoir serves two main purposes. Firstly, it helps to determine when spectra should be refined, and secondly, it provides a cache for extracting the class label whenever several data instances recur within the same interval with the same schema, or when schema only differ in one or more index positions involving non contributing features. In this respect the higher the number of collisions in the hash table, the better the efficiency. For any given schema the class value returned by the IFT will remain the same and will represent the predicted class value as long as the spectrum remains unchanged.

5.3. Algorithm

The pseudocode given in algorithm 1 presents the reservoir management and classification process.

As given in steps 4-9, if the hash value returned by the corresponding spectrum's hash function already exists in that spectrum's reservoir, instead of reclassifying it we simply extract the class value from its first appearance. If no entry exists, then it is necessary to classify and insert a new record. If the winner is not in a drift state and a threshold I (set to 100, by default) on the number of instances arrived (steps 10-12) then the Refine Winner procedure is activated.

In algorithm 2, the requirements for refinement are first tested in step 6. In case the cumulative change rate is higher than the tolerable change rate T (set to 0.05 by default) the coefficients of the winner spectrum will be modified by utilizing an incremental factor, as shown in step 15 of Algorithm 2. At this point in time the reservoir is reset by clearing its contents (step 15 of Algorithm 1).

If it fails to meet the conditions for refinement, subsequent tests for refinement will be done using the accumulated change rate in intervals of size I.

Algorithm 1: Incremental Fourier Classifier

```
Data: Interval Size=100, Spectrum Repository, InstanceCount =0 , EffectiveChangeRate=0
while InStage2 do
    Read Instance ;
    Increment Instance Count by 1;
    foreach Spectrum S_i in Repository do
        Apply S_i's Hash Function on Instance;
        if Hash Value found in S_i's Reservoir then
            Extract predicted class value, update frequency and True class columns ;
        else
            Classify by IFT and insert into Reservoir;
    if Winner is not in drift then
        if Instance Count MOD Interval Size = 0 then
            EffectiveChangeRate= RefineWinner (EffectiveChangeRate);
            if WinnerRefined then
                Reset Instance Count =0;
                Clear Winner's Reservoir ;
            else
                foreach Winner Reservoir Record do
                    Previous True Class = Current True Class;
                    Current True Class =null , Frequency =0 ;
    else
        Define spectrum with highest accuracy as New Winner;
        Aggregate Previous Winner with most similar Spectrum if such exist;
        Reset InstanceCount =0 ,Reset EffectiveChangeRate=0;
        Clear New Winner Spectrum's Reservoir;
```

6. Empirical Study

The empirical study examines the accuracy and throughput of IFC against a selection of 11 other algorithms that have been proposed for concept drifting data streams. These algorithms were chosen carefully: 9 of them are state of the art meta learning algorithms featured in MOA and the other two (SOL and RC) use the staged approach as IFC does but with differences in the learning approach employed.

We start by describing the datasets and algorithms used in the study and then go on to conduct a comparative analysis on both accuracy and throughput.

6.1. Datasets Used for the Empirical Study

All experiments were carried out with the use of two synthetic datasets generated by MOA's stream generators[1] and five real world datasets.

Synthetic Data Recurring with Noise:
For synthetic datasets we generated several distinct concepts, each of length 10000 instances. With the belief that concepts do not repeat in exactly the same form in reality, we introduced a 5% noise level for each concept recurrence by flipping the class label of randomly selected data instances.

1. Rotating Hyperplane dataset (RH): This dataset has a total of 10 attributes and we adjusted the magnitude of change in the range [0.03, 0.04, 0.05, 0.07, 0.08, 0.09] to generate 5 distinct concepts. The first three concepts were repeated 20 times, with each concept being distorted by a noise level of 5% at each cycle over is base representation (i.e its first generated state). Three previously unseen concepts were injected at the end of datasets with the intention of examining whether the staged learners would opt for adapting to the new concepts in Stage 2 or for triggering T_2 to transit back to Stage 1. The size of the dataset was 660,000 data instances.

2. RBF dataset (RBF): This 10 dimensional dataset generated concepts by changing the number of centroids. The size of this dataset is 1,300,000 data instances with 5 different concepts which were repeated 25 more times with noise as per the description for RH.

[1] from *http://moa.cms.waikato.ac.nz/*

Algorithm 2: Refine Winner

Data: Tolerable Change Rate=0.05, Winner Refined = False, Number of Changes =0;
foreach *Winner Reservoir Hash Value appeared in Current Interval* **do**
  if *Current True Class is not equal to Previous True Class* **then**
    Increment $NumberOfChanges$ by Frequency;
$ChangeRate = NumberOfChanges/IntervalSize$;
$CumChangeRate = EffectiveChangeRate + ChangeRate$;
if *CumChangeRate > Tolerable Change Rate* **then**
  foreach *Coefficient Index j that belongs to Winner Spectrum* **do**
    $IncrementalFactor = 0$;
    foreach *Reservoir Hash Value x where Classes are changed* **do**
      Calculate $ClassDifference = (f_{(x)}^{current} - f_{(x)}^{previous})$;
      Compute $F = \psi_j(x) \times ClassDifference$ where $\psi_j(x)$ is from Eq. 1
      // Update Incremental Factor in Eq. 9 as $\sum_{x \in C} \psi_{\vec{j}}(x)(f^{current}(x) - f^{previous}(x))$
      $IncrementalFactor = IncrementalFactor +$ F;
    Compute $AverageIncrementalFactor$ as $\frac{1}{|X|} IncrementalFactor$; //where $|X|$ is the current reservoir size
    $w_j^{new} = w_j^{current} + AverageIncrementalFactor$ // as in in Eq. 9
  Set $WinnerRefined = True$, Reset $CumChangeRate = 0$;
else
  Set $WinnerRefined = False$;
return $CumChangeRate$;

Real World Data: The real word datasets that we experimented with have been widely used as benchmarks in a number of studies on data stream mining. The accuracy profiles of the most accurate classifiers show that these datasets have widely different levels of concept drift and concept recurrence. The Flight dataset was generated by NASA's FLTz flight simulator and was obtained from the Nasa website [2]. This has total of 30 features and 25,030 data instances. We experimented with two bin discretized form of the NSW Electricity(Elec) dataset[3].The Covertype(Cover) original dataset is available at Lichman (2013). We extracted instances from the two most frequent classes and used a two bin discretized form for the features. The dataset and description of Occupancy(Occ) is found from Lichman (2013) and previously experimented by Candanedo & Feldheim (2016). The Sensor stream (Sensor) dataset extracted from Zhu (2010) contains a total of 130,073 instances.

6.2. Algorithms used in study

Nine of the twelve algorithms that we experimented with were sourced from MOA[4] under its meta learning category and were explicitly designed for time changing data streams. We also used an approach proposed in Kithulgoda & Pears (2016) that used the staged approach but used decision trees as its learning mechanism instead of Fourier spectra. This algorithm is termed Staged Online Learning (SOL) as it uses a staged approach. In Stage 2 it reconstructs trees from a pool of Fourier spectra and uses the trees for learning and classification. This algorithm was selected as it would provide an interesting contrast between Fourier learning and classification against the use of a conventional decision tree classifier.

We also experimented with another implementation of the Staged approach which simply uses Fourier spectra generated in Stage 1 to classify data arriving in Stage 2 without any form of learning. This classifier thus assumes recurrence of concepts in Stage 2 and is thus termed Recurrent Classifier or RC. This version provides a useful contrast with the Incremental Fourier Classifier (IFC) proposed in this research as it enables us to assess the benefits of adapting spectra in Stage 2.

The nine meta learning algorithms derived from MOA are: Adaptive Random Forest (ARF)Gomes et al. (2017), OzaBagASHT (AS), OzaBagADWIN (OB)Bifet et al. (2009), LeveragingBag (LB) Bifet et al. (2010b), LimAttClassifier (LA) Bifet et al. (2010a), AccuracyWeightedEnsemble (AWE) Wang et al. (2003), AccuracyUpdatedEnsemble (AUE) Brzeziński & Stefanowski (2011), Anticipative Dynamic Adaptation to Concept Change (AD) Jaber et al. (2013a) and Dynamic Adaptation to Concept Changes(DA) Jaber et al. (2013b).

[2] from *https://c3.nasa.gov/dashlink/resources/*
[3] https://moa.cms.waikato.ac.nz/datasets/
[4] from *http://moa.cms.waikato.ac.nz/*

Table 2: Classification Accuracy with ranking

	RBF	RH	Flight	Elec	Cover	Occ	Sensor	Algorithm Rank
ARF	81.8(2)	73.9(4)	80.8(2)	**68.5**(1)	84.5(3)	**95.9**(1)	65.2(6)	**2**
OB	75.3(10)	73.7(5)	73.0(6)	65.4(4)	83.2(5)	82.3(9)	66.5(3)	5
AS	80.3(4)	74.2(3)	73.1(5)	64.4(5)	75.6(9)	84.0(8)	62.2(9)	6
LA	74.8(11)	75.3(2)	71.4(9)	61.3(11)	84.5(3)	89.1(5)	62.9(8)	7
LB	**82.4**(1)	**76.3**(1)	77.2(3)	66.0(3)	**86.1**(1)	86.4(6)	66.2(4)	**2**
AWE	72.1(12)	69.3(9)	49.3(10)	63.7(9)	78.5(8)	73.4(12)	65.8(5)	10
AUE	76.4(7)	71.1(7)	72.5(7)	64.2(7)	82.8(7)	81.8(10)	65.1(7)	8
AD	76.2(8)	68.9(10)	76.9(6)	64.3(6)	84.3(4)	95.3(3)	65.8(5)	5
DA	75.9(9)	68.9(10)	77.2(3)	64.0(8)	84.5(3)	95.8(2)	65.8(5)	4
IFC	80.7(3)	74.2(3)	**81.8**(1)	**68.5**(1)	84.6(2)	93.0(4)	**70.3**(1)	**1**
RC	79.0(5)	70.6(8)	71.6(10)	62.4(10)	73.8(10)	75.6(11)	54.4(10)	9
SOL	78.7(6)	72.7(6)	80.8(2)	67.2(2)	82.9(6)	85.4(7)	69.5(2)	**3**

6.3. Parameter Values

The default parameter values used in the experimentation are as follows:
Maximum number of nodes in decision tree forest:5000, concept drift significance confidence (δ)=0.01, Maximum number of Fourier spectra in repository:20, Repository hit ratio threshold α for T_1 is 0.5. In IFC, the tolerable change rate (T) for refinement is 0.05 . Refinement interval length (I) is 100.

6.4. Accuracy Evaluation

In order to maintain fairness all classifiers were run with the Hoeffding tree as the base learner. The split confidence and tie confidence parameters were both set at 0.01 for all classifier ensembles. Each meta learner was run with default settings for its internal parameters. Accuracies for each classifier was obtained in intervals of size 1000 and an overall accuracy was then computed by averaging over all intervals for a given dataset. Accuracies were stable across multiple runs for all classifiers across all datasets with a standard deviation less than 0.05 and hence were not included. The outright and joint winners (with accuracies rounded to 1 significant place) were bolded for easy identification of winner. Ranks for each algorithm were included in parentheses in order to facilitate the analysis of the trade-off between accuracy and throughput which we conduct in Section 6.6.

Table 2 shows that the algorithms can be divided into 3 groups with respect to accuracy. The first group of the most accurate classifiers consisted of ARF, LB, IFC and SOL. Interestingly, the accuracy ranks (shown in parentheses) of three out of four of these classifiers indicate that they collectively accounted for winners across the entire range of datasets. The accuracy profiles of LB and ARF reinforce results from previous studies where they excelled in performance Bifet et al. (2010b),Gomes et al. (2017). The strong performance of ARF is to be expected as the basic Random Forest algorithm has proved its superior performance in numerous studies and ARF builds on this success by adapting the forest to concept drift in a data stream environment.

The middle group comprised of AD, DA and OB while the rest formed the bottom group. We further analyze the performance of the algorithms using RC as a benchmark. Since RC does not adapt its model upon entering its low volatile state (Stage 2 of its execution), it is an ideal algorithm to assess the impact of the scale of concept change on the performance of the algorithms. Using this benchmark, we observe that the Sensor, Occ, Cover and Flight datasets gave rise to the highest accuracy improvements with respect to RC.

The IFC execution log with the Sensor dataset revealed moderate rates of drift and refinement when compared to the other datasets. IFC gracefully copes with moderate intra-concept variation by incremental refinements on its initially captured spectra while inter-concept switches trigger concept drift and re-use of previously captured spectra. In contrast, LB had to re-learn concepts from scratch rather than making slight adjustments or reuse of previous concepts. The effects of this on LB's accuracy profile is apparent from Fig. 5 as there is a noticeable time lag between its peaks and those of IFC over much of the stream. IFC is able to capitalize on concept recurrence by self adaptation or by replacing its currently used spectrum with another in its pool that more closely matches the concept that has recurred. Also the 29% of accuracy advancement with respect to RC highlights the importance of being self adaptive through periodic refinements of initially learned patterns.

Similarly, the Occupancy dataset also revealed significant improvement (23%) of accuracy over RC and 8% improvement over LB. The execution log of IFC revealed that Occupancy has a high rate of inter-concept variation, resulting in switches among a few distinct concepts which were captured by only 5 different spectra in the repository. At the same time it demanded a high rate of adjustments on recurring concepts thus strengthening the need for refinements to be made on models stored from the past.

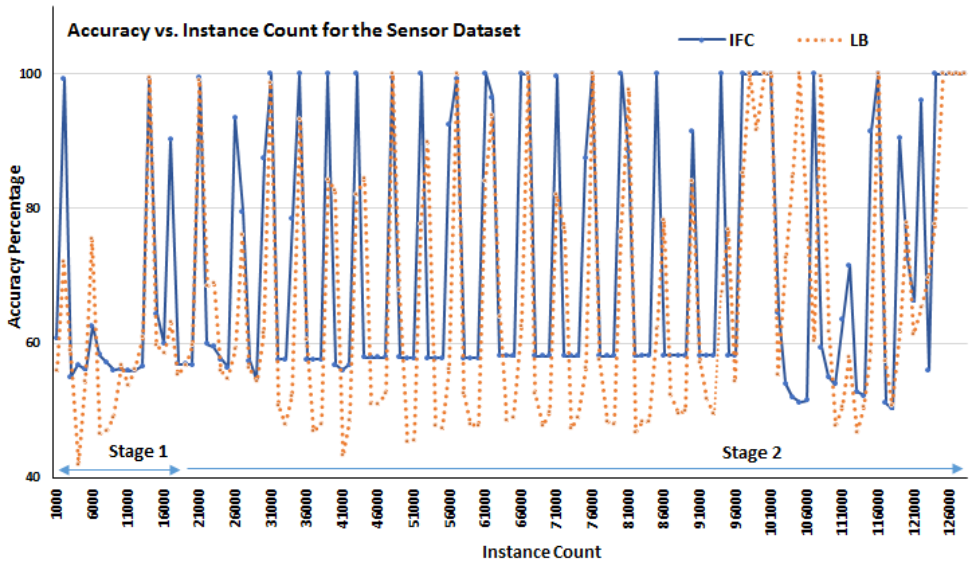


Figure 5: Sensor Accuracy Chart

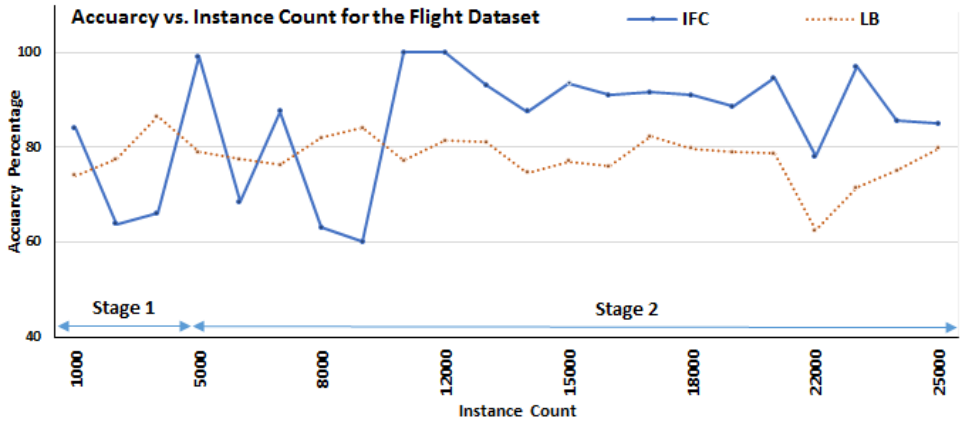


Figure 6: Flight Accuracy Chart

On the other hand LB excelled on the Cover dataset. The accuracy profiles of these classifiers over time show no regular pattern in the peaks and troughs, suggesting a lack of significant recurrence of concepts over time for this particular dataset. The LB adapts by adjusting the trees in its ensemble to new concepts as they occur. The IFC classifier experienced its highest rate of refinement amongst all datasets, resulting in significantly different spectra through the refinement process in Stage 2. This scenario exemplifies high inter-concept variation between concepts which are interspersed with abrupt intra-concept changes within a given concept. Even though the performance of IFC is somewhat lower compared to LB, we note that IFC is second best with a 15% of improvement over RC. This shows IFC's ability to provide good accuracy even in non-recurring situations by generating models that are derivatives of previously stored ones.

IFC with the Flight dataset experienced its highest inter-concept drift rate. The number of concepts in its repository is lesser when compared to Cover but higher when compared to Occupancy. IFC obtains best accuracy for this dataset by taking advantage of reuse of concepts but also by subjecting them to continuous refinement. The accuracy advancements are 14% and 6% over RC and LB respectively. As shown in Fig. 6 IFC remains the clear winner in Stage 2 of its execution cycle as a result of reuse and refine of concepts, whereas with LB concepts were relearned.

In terms of RBF, LB was the winner with marginally better accuracy than IFC. LB adapted its model well to concept drift in this dataset. A close analysis of the models produced showed that LB produced the deepest trees in its ensemble, with comparatively higher concept lengths when compared to other datasets (with the exception of

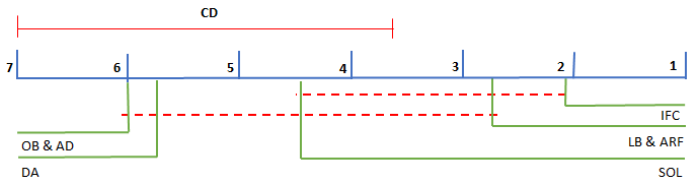


Figure 7: Statistical comparison of top 7 ranked algorithms by accuracy. Subsets of classifiers that are not significantly different (at the 95% confidence level) are connected with dashed lines.

RH). Such deep trees when grown over a significant time duration do better than IFC and other algorithms, especially when drift rate is low. With IFC the comparison was harder in terms of tree depth as it relies exclusively on Fourier spectra in Stage 2 but its average tree height in Stage 1 was much shorter than that of LB.

Finally, we note that the triad of algorithms, ARF, IFC and LB performed well across the full range of datasets, in general. The SOL classifier which is at the thrid place in overall performance also remains competitive with IFC on Flight, Elec, and Sensor datasets. The two classifiers share the use of the DFT although IFC in Stage 2 relies entirely on Fourier spectra whereas SOL uses Hoeffding trees induced from spectra as the learning mechanism. The essential difference between the two is that a Hoeffding tree induced from a spectrum is constrained by its initial structure. The tree is free to grow by splitting its leaf nodes and forming new decision nodes but changes in higher level nodes are not possible. In contrast, Fourier spectra are refined when changes occur in the stream and such changes could reflect more fundamental changes in its decision model. Conceptually, such changes could correspond to better adaptability to changes that take place over time in the stream. In other words spectra have better flexibility and are able to prune away redundant decision paths and replace them with new ones that are a better match with currently arriving trends in the stream.

The above analysis of the behaviour and performance of classifiers was based on the groupings formed by considering average accuracy rank on the datasets as shown in Table 2. This grouping was confirmed through a non-parametric Friedman test that established statistically significant differences between certain sets of classifiers. For this exercise we subjected the top 7 algorithms by average accuracy rank to the Friedman test. As discussed in Demšar (2006) the Friedman test is recognised as one of the best tests to use when multiple classifiers are to be compared simultaneously on multiple different datasets. The null hypothesis H_0 that we used was that the average accuracy ranks across the 7 classifiers was the same.

Fig. 7 shows that the null hypothesis H_0 was rejected at the 95% confidence level, thus indicating that statistically significant differences exist between the classifiers. We then subjected the classifiers to the post hoc Nemenyi test to identify exactly where those differences lay. The Nemenyi test yielded a critical difference(CD) value of 3.4 at the 95% confidence level. Fig. 7 graphically illustrates that the top group consisted of IFC, LB, ARF and SOL as none of the members in this group had significant differences with any of the other members within the group. All members of the other group (i.e. OB, DA and DA) are also not significant from other members of their group. However, we can also note from Fig. 7 that the IFC classifier stands out as it is the only classifier that was significantly different from all other members of the other group. This was not the case with ARF, LB and SOL. For instance, LB, ARF and SOL are not different from OB, AD and DA.

6.5. Throughput Evaluation

We now turn our attention to the processing speed of the algorithms, and in particular we examine how IFC fares in relation to the other 11 algorithms we use. We measure processing speed over the same interval size (1000) just as we did with accuracy. Processing speed is measured by throughout which is the number of data instances processed in an interval for any given dataset. An average throughput measure was then computed over the entire dataset. We conducted multiple runs for each dataset and observed negligible variance in timing between runs. The performance study was conducted on a machine having 16GB of RAM, featuring two dual core processors rated at 3.2 GhZ running under Windows 10.

For the purpose of simulating real world data streams we increased the size of the smaller datasets, namely Elec, Flight, Cover and Occupancy by concatenating each of them with multiple copies of themselves. This also served to produce more reliable estimates of execution time and throughput. The other 3 datasets were large enough to be used in their original form.

Although IFC performed well in terms of accuracy it was designed specifically to scale up to high speed data streams and the performance aspect is crucial in its evaluation. Table 3 shows that IFC excels on throughput, obtaining the highest average rank when taken over all datasets and emerging the outright winner in 3 out of the 7 datasets.

To gain insights into why IFC was the overall winner we compare its throughput to that of the RC, SOL and the MOA group of algorithms. We first compare IFC with SOL and RC as they all use the staged approach. We

Table 3: Throughput with ranking

	RBF	RH	Flight	Elec	Cover	Occ	Sensor	Algorithm Rank
ARF	25811(8)	26481(7)	18866(3)	33038(7)	11837(4)	45729(9)	50342(7)	6
OB	30729(4)	30634(5)	4975(7)	37313(6)	4398(6)	51692(8)	50263(8)	5
AS	30056(5)	39910(4)	6353(6)	53882(5)	3371(8)	71425(3)	86447(2)	4
LA	3195(12)	2827(12)	1686(11)	25869(10)	117(12)	64000(4)	64000(3)	8
LB	9810(9)	10838(11)	1082(12)	21667(12)	1413(11)	33641(12)	36525(11)	11
AW	**59468**(1)	47099(3)	19722(2)	54428(4)	14122(3)	60618(6)	62345(4)	2
AU	50577(2)	59704(2)	10777(4)	55755(3)	6811(5)	63289(5)	62080(5)	3
AD	26446(7)	24933(8)	3517(9)	23110(11)	2788(10)	39282(11)	43313(10)	10
DA	29028(6)	24138(9)	3720(8)	30501(9)	2883(9)	45630(10)	53818(6)	7
IFC	35599(3)	**73335**(1)	**70990**(1)	89338(2)	35466(2)	155560(2)	**137466**(1)	1
RC	9468(10)	26553(6)	7975(5)	**415660**(1)	**83856**(1)	**464493**(1)	48889(9)	4
SOL	5140(11)	17076(10)	3044(10)	31106(8)	4102(7)	53031(7)	30869(12)	9

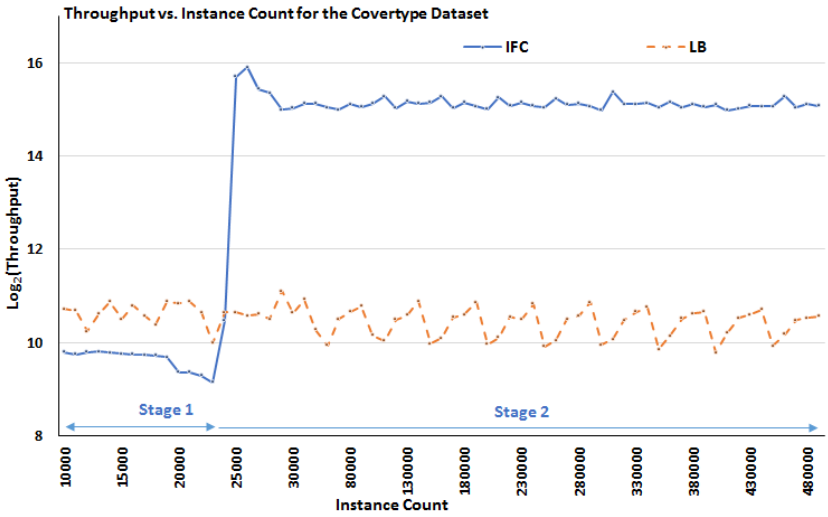


Figure 8: Covertype Throughput Chart

observe that IFC significantly outperforms RC on RBF, RH, Flight and Sensor datasets. Despite the fact that RC does not update its spectra during Stage 2, IFC is much faster due to the lesser number of distinct schema when compared to RC. RC does not use feature selection and the number of influential features in dot product calculations are much higher than that of IFC. Since it uses the inverse Fourier transform in Stage 2 which has complexity $O(|X|^2)$, its Stage 2 processing time is much longer than that of IFC.

In contrast, SOL uses trees induced from spectra during Stage 2. Even though it maintains a single tree at a time instead of an ensemble it still has higher classification overhead than IFC. The IFC classifier simply fetches the class label of an instance to be classified from its reservoir if present, otherwise it performs the inverse Fourier transform but unlike RC its schema set is much smaller and hence the overall classification time is much smaller.

In comparison to IFC, all of MOA's classifiers use an ensemble of trees during Stage 2, and depending on average tree size their performance is impacted. In particular we see that LB that was ranked alongside ARF and IFC on accuracy suffers the most when compared to IFC as it had the highest tree size out of all of MOA's group of classifiers. Overall, the ARF classifier had a higher throughput than LB even without the use of a large multi core machine for which it was designed. It was reported in Gomes et al. (2017) that its speed increased by a factor of around 3 with the use of a machine that supported a high degree of parallelism with 40 cores.

Figures 8 and 9 show the contrast between the IFC and LB classifiers. In Fig. 9 the granularity of the horizontal axis in the range 1000 to 10000 was scaled up in order to highlight the presence of the relatively short lived Stage 1. These charts clearly show that IFC's speed advantage was gained in Stage 2 of its processing when classification was performed exclusively with the use of Fourier spectra.

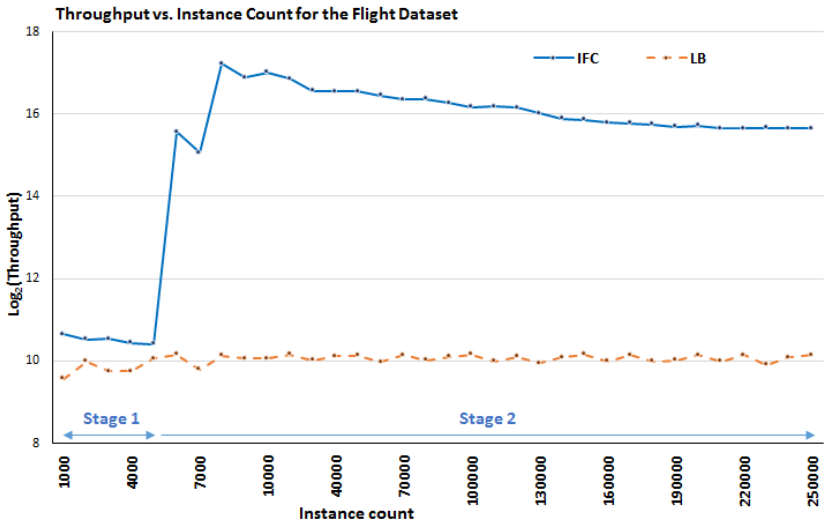


Figure 9: Flight Throughput Chart

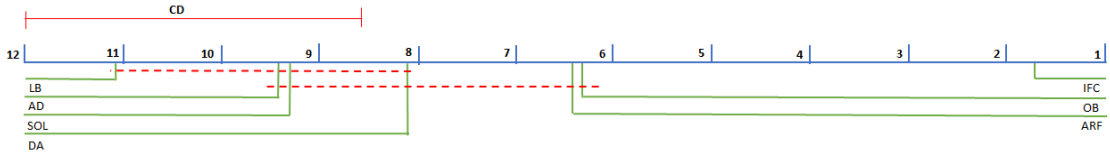


Figure 10: Statistical comparison of the throughput performance of the top 7 ranked algorithms that were ranked on accuracy. Subsets of classifiers that are not significantly different (at the 95% confidence level) are connected with dashed lines.

In order to gain an understanding of how the most accurate classifiers perform with respect to throughput we subjected the same set of 7 classifiers that we selected for accuracy analysis to the Friedman test. Fig. 10 shows that there are three distinct groups. The first group comprises of IFC on its own as it has statistically significant differences with all other 6 classifiers that form the other two groups. The second group consists of ARF, OB, DA and AD while LB makes up the another group together with AD, SOL and DA . This statistical analysis thus yields additional insights into the relative throughput performance of the classifiers over those given in Table 3.

6.6. Accuracy versus Throughput trade-off

We now visualize the trade-off between accuracy and speed. Fig. 11 clearly shows that IFC and LB are at opposite sides of the speed spectrum. Classifiers that tend to be more accurate (e.g. LB, SOL) tend to be more time consuming and vice versa; the exception being IFC although it was not always the most accurate, nor was it always the fastest. However, it is clear that IFC has achieved its design goals as it is by far the fastest amongst the set of the most accurate algorithms that we experimented with. The closest neighbours to IFC are ARF and OB as they achieved the next best balance between accuracy and speed.

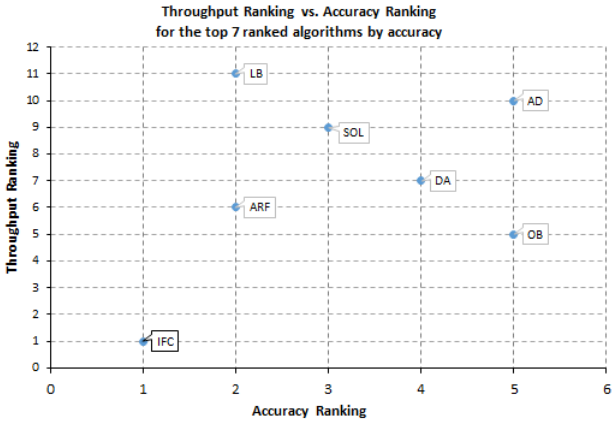


Figure 11: Accuracy vs. Throughput tradeoff

6.7. Load Shedding

Finally, we subjected the IFC and LB classifiers to a load shedding exercise that assessed their sensitivity to accuracy by simulating a high speed data stream environment.

Firstly, we used the speed achieved by IFC as the benchmark and then manipulated parameters for LB until it matched IFC's throughput. The matching was achieved by decreasing the number of classifiers in LB's ensemble

and/or setting the split confidence parameter value for the Hoeffding tree base classifier to low settings (e.g. 10^{-7}). The load shedding simulation was performed for the RBF, Flight and Cover datasets. These datasets were chosen on the basis of their accuracy performance with the LB and IFC classifiers and the fact that they represented the greatest diversity in dimensionality across the datasets that we experimented with.

Table 4 when read in conjunction with Table 2 shows clearly that IFC now achieves the highest accuracy over each of these datasets. This result show that an algorithm's accuracy needs to be judged by its load handling capability - while it may be more accurate than another algorithm on a stream with a certain speed its accuracy advantage may disappear when it runs on a higher speed stream and is forced to perform load shedding to cope with the speed of the stream.

We also tracked IFC's ability to scale up to data streams in excess of its capacity reported in Table 3. We simulated load shedding with IFC by increasing its interval size I by a factor of 5. Table 5 shows that IFC's throughput gain ranged from a modest 13.3% for the Flight dataset to 55.6% for the Cover dataset. Interestingly, we see that the accuracy for the RBF dataset increased marginally by 0.5%. This is consistent with the moderate drift rate detected in IFC's execution log for this dataset. The decrease was 3.8% for Cover and 5.6%. The Flight dataset recorded the highest decrease as it exhibited the highest drift rate and highest rate of refinements amongst all datasets.

Table 4: Load Shedded LB Accuracy

Dataset	Ensemble Size	Split Confidence	Accuracy
RBF	7	1E(-4)	80.0
Flight	3	1E(-7)	73.6
Cover	3	1E(-7)	82.4

Table 5: Load Shedded Throughput and Accuracy for IFC

Dataset	Scaled Throughput	Throughput Gain (%)	Accuracy
RBF	45156	26.8	81.1
Flight	80408	13.3	77.2
Cover	55155	55.6	81.4

6.8. Overheads of Decision Tree learning vs Incremental Fourier Classification

In this section we contrast the overheads of classification with a decision tree to an incremental learning approach using Fourier spectra. The Staged Online Learner (SOL) that we featured in our experimental study uses the staged learning approach just as IFC does, except that it classifies using a decision tree instead of a Fourier spectrum in the low volatility segments of the stream. The SOL classifier takes advantage of low volatility to dispense with a forest of trees and uses a single tree instead. This provides a good platform for comparison as IFC which uses an ensemble of models (in the form of spectra) would need to compete with a single model.

Fig. 12 depicts the time spent by IFC and SOL in randomly selected stream blocks. The blocks were selected from the execution logs of both classifiers to exclude blocks where either SOL and/or IFC experienced concept drifts. This was done to ensure that the time reported reflected the time spent in classification. The overheads of processing concept drift are presented in our study in Section 6.9.

Fig. 12 shows clearly that classification using Fourier spectra is much faster than with a Decision Tree. This is due to the replacement of a tree traversal operation with a simple look up operation on the hash based reservoir structure that extracts class labels in the event of a cache hit. In the event of a cache miss the inverse Fourier transform would be used to retrieve the class label of the instance.

6.9. Incremental versus Non Incremental Approach to Fourier Classification

In this section we study the effects of incrementally maintaining a Fourier spectrum and contrast it with a naive version which generates a new spectrum from a decision tree whenever a concept drift is signalled in the stream.

We conducted two new experiments. The first was on the Flight dataset and shows the difference in timing between the two versions at concept drift points. It is clear from Fig. 13 that the incremental version is far more efficient and is able to avoid spikes in processing time that are experienced by a conventional DFT application.

The second experiment was a controlled experiment run on a RBF dataset whereby we varied the dimensionality from 10 to 60 in intervals of 10. The intention was to examine the CPU utilization as a function of dimensionality (D), drift rate (R) and stream speed (S). For each value of dimensionality we recorded the average processing time spent on classifying data instances (C) as well as the time spent (F) on the application of the DFT. Likewise, with the incremental we recorded the average time spent in classification as well as in reorganizing the spectrum with our incremental DFT version. With these average values in place we modelled CPU utilization (U) using Eq. 11.

$$U = (1 - R) \times S \times C + R \times S \times F \tag{11}$$

We first model a stream with a sample arrival rate of 1000 instances per second. The top panel of Figure 14 shows clearly that CPU utilization is an increasing function of both drift rate and dimensionality for both the incremental and non incremental versions. For the non incremental version in the top left panel of Figure 14 we note

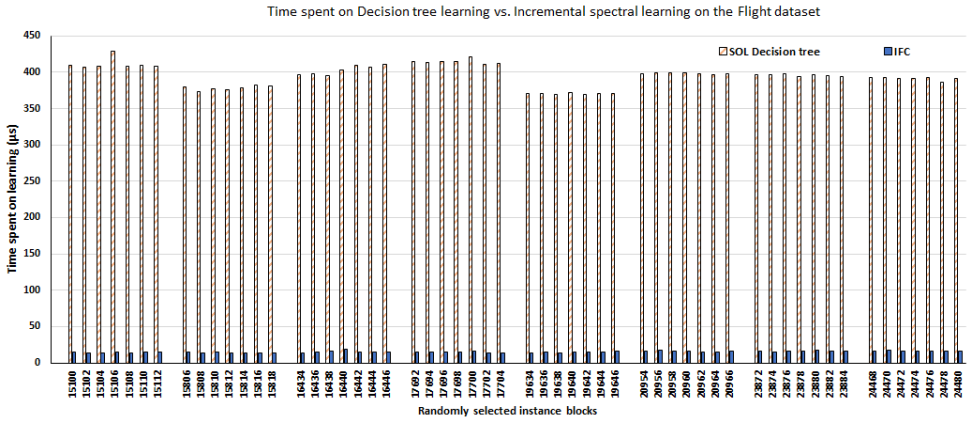


Figure 12: Throughput advantage of Incremental spectral learning over Decision tree learning

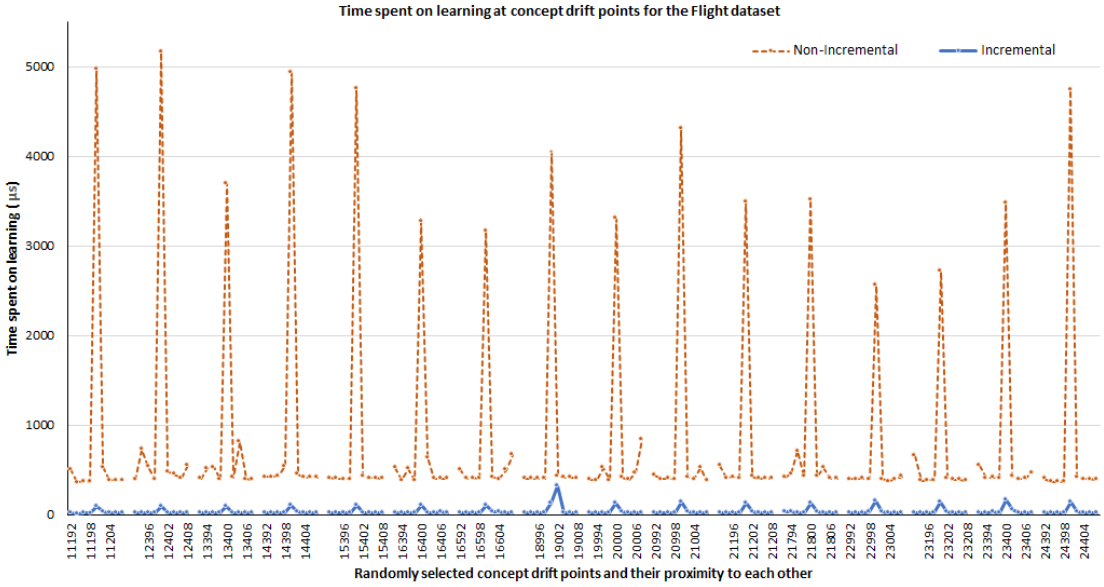


Figure 13: Time spent when learning in Non-Incremental and Incremental modes of operation

that the utilization varies from a low bound in the range 0-20 to an upper bound of 120-140 when both dimensionality and drift rate are their maximum levels. At the high end of the CPU range (120-140), 2 resources with own processor and local memory would be needed to ensure that the system functions without any form of disruption.

In sharp contrast we note from the top right panel of Figure 14 that the upper bound for CPU utilization is very much smaller at 2.5%, thus managing comfortably with a single resource.

We next model a high speed data stream with a speed setting of 100,000 instances per second. The bottom left and right panels of Figure 14 shows the same trends as with a stream speed of 1000 but at a much higher scale. With the non incremental version, the left panel of Figure 14 shows that the CPU utilization reaches a maximum value of 12000-14000, thus requiring up to 140 resources to maintain operation without disruption. If lesser than 140 resources are available then the cost of the non incremental version becomes prohibitive. In contrast, the peak CPU utilization for the incremental version was just 300-350; thus requiring far fewer resources (up to 4) to maintain normal operation.

These experiments show clearly the resource utilization advantage of the incremental approach and explains the massive gains in throughput of IFC over the non incremental SOL classifier that we tabulated in Table 3.

7. Application of the Fourier Transform to other domains

In all of the above discussions we have presented the use of the Fourier Transform in a supervised learning context. In fact, the DFT can also be used in a data engi-

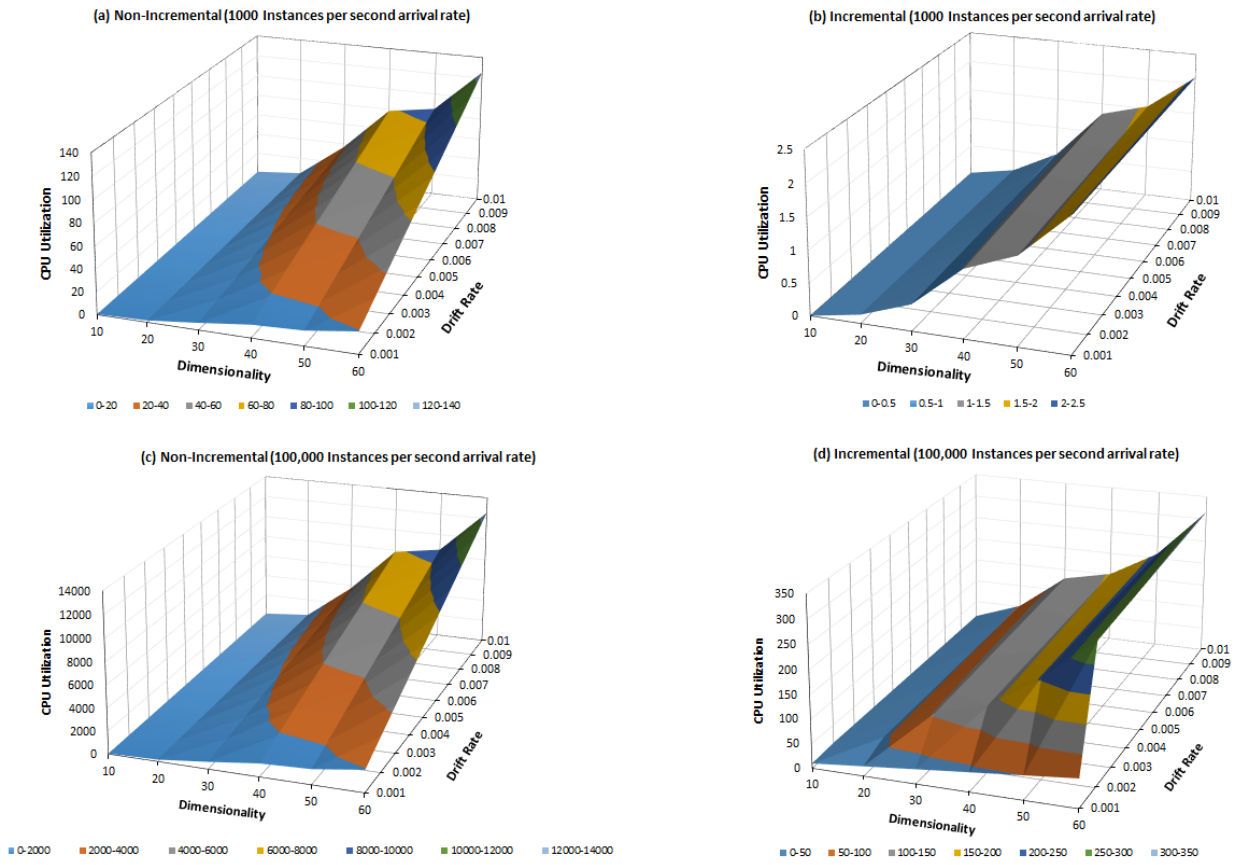


Figure 14: Resource Utilization of Incremental and Non-Incremental DFT approaches

neering context. An important spatiotemporal problem is to keep track of the movements of people in both space and time simultaneously. An efficient approach for solving this problem was reported in Moreira-Matias et al. (2016) where an OD matrix was used to capture the origin and destination points of travellers within a specified spatial boundary.

The OD matrix can be expressed through a spatial index such as a k-d tree Bentley (1975).

The k value denotes the number of keys used to index paths in the tree. With a k value of 2 the spatial index can be represented as a binary tree. The root node of the tree represents the entirety of the area being mapped. Each subsequent node represents a decision node or a leaf node. Decision nodes represent geographical regions which are subdivided into two smaller regions until further subdivision would violate some user-defined constraint on the density of people in a subdivision or on geographical size. A discriminator variable is associated with each level of the tree and indicates whether the first or second keys are used for splitting a decision node into its child nodes. The discriminator variable alternates in value between levels, starting from a value of 0 at the root level of the tree.

A concrete application of such a tree would be to track the movements of people across different destinations given a point of origin. In this case the root node would represent the point of origin and leaf nodes would represent destinations. The decision nodes represent geographical regions on the course of a journey that ultimately lead to destination points.

Fig. 15 illustrates an example spatial grid with 4 splits that define 5 points that would be potential destinations (D_1, D_2, D_3, D_4, D_5) for travellers. Leaf nodes hold information on the relative popularity of destinations. Thus for example, the leaf node D_1 represents the ratio of the number of travellers who have journeyed from X_1 to D_1 to the total number of travellers proceeding from origin X_1, thus yielding a ratio value between 0 and 1. With a user defined number of bins (n), the popularity variable can be discretized into a set of categorical values lying between 0 and n-1. Using this representation and the notation presented in Section 2, each tree path can be encoded as a schema instance together with its popularity value.

The schema set would consist of 16 schema instances in total. We observe that multiple schema instances refer to the same path. Thus for example the path from X_1 to destination D_1 is represented by 0000 and 0010 as feature X_3 is not a discriminatory feature on this path. With the use of a wildcard character $*$ which takes values of either 0 or 1 the number of schema instances to be stored is deter-

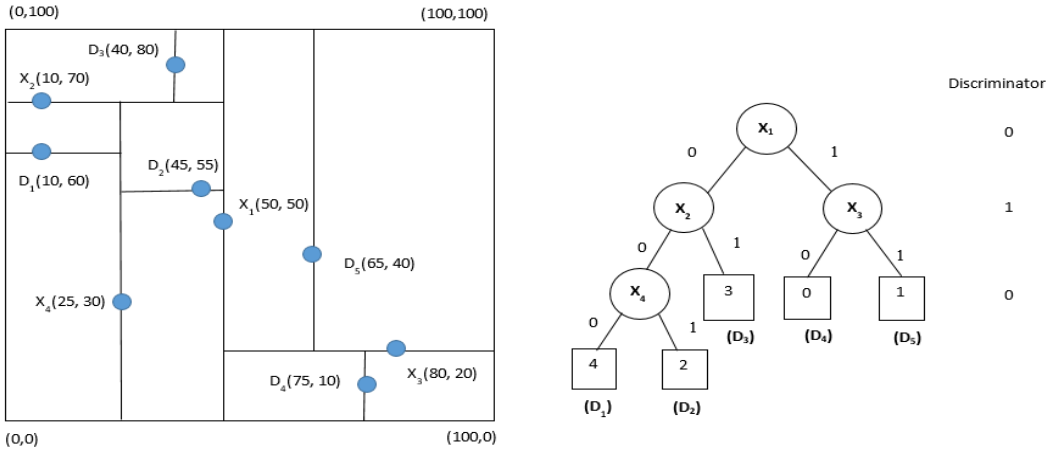


Figure 15: An example spatial grid and its k-d tree representation for an O-D matrix application

Table 6: Schema for Fourier encoding of k-d tree

Destination Key	X_1	X_2	X_3	X_4	Popularity
D_1	0	0	*	0	4
D_2	0	0	*	1	2
D_3	0	1	*	*	3
D_4	1	*	0	*	0
D_5	1	*	1	*	1

mined by the number of destinations. This representation is illustrated in Table 6 and reduces the number of schema instances to be stored substantially, which in our example entails a reduction from 16 to 5. The compact schema representation in Table 6 can be used to generate the Fourier spectrum.

Once the spectrum is generated, a query such as $Q(X_1 \rightarrow D_4)$ can be answered by matching the destination string in the consequent portion of the query to the destination value in the schema table and then extracting the corresponding row with its schema instance. The extracted schema, which happens to be $1 * 0 *$ is then subject to the inverse Fourier transform to compute its popularity value. In general, a schema instance $\vec{x}$ extracted from the schema table would be subject to equation 2 to yield the popularity value $f(\vec{x})$.

As we have seen from the experimentation presented in Section 6, the advantage of encoding search trees by Fourier spectra is that expensive tree searches and maintenance can be replaced by efficient algebraic operations. This example illustrates that the utility of the Fourier application to trees is not restricted to supervised learning.

8. Conclusions and Future Work

In this research we presented a novel approach to classification that exploits the Discrete Fourier transform. The Fourier Classifier although designed primarily for speed, outperformed a collection of widely used data stream classifiers on both the accuracy and speed dimensions. Our empirical study revealed that IFC was able to adapt its behaviour successfully to different types of situations that manifest in data streams.

In stream segments exhibiting recurrence of previous concepts it re-used previously stored spectra that most closely matched with the recurring concept. On the other hand when the current stream segment did not closely resemble any of its stored concepts from the past it refined the spectrum that it currently operated on to adapt to concept change in the stream. We expect most data steams to be hybrid in nature, exhibiting both recurrence and concept change characteristics and hence IFC's learning mechanism will be ideally placed to take advantage of these type of environments.

In terms of speed, IFC proved to be superior on account of its compact data structure used for classification. The coefficient pruning strategy implemented via energy thresholding ensured that the coefficient array was as compact as possible, thus reducing classification overhead. In addition, the schema pruning strategy, when used in conjunction with the incremental Fourier coefficient update strategy ensured that update overheads were minimized.

In terms of future work there are two main directions for further optimization of throughput. Firstly, there is scope for performing a more aggressive feature selection strategy than the one we proposed in Section 5.1. Currently, features are considered to be redundant if they do not appear in a decision tree. However, it is possible to exclude features that do appear but still make a minimal contribution to classification accuracy. This will be especially useful in high dimensional data streams.

Another direction that future work can proceed is in exploiting parallelism. Unlike the classical decision tree, IFC is well placed to exploit parallelism in the classification process. Classification via a decision tree is inherently a sequential process. Classification of a newly arriving data

instance would require traversing the tree from its root to its target leaf node which is essentially a sequential search process. On the other hand, classification with IFC can be done by partitioning the Fourier array into a number of segments and performing the Inverse Fourier transform across the different segments in parallel. We plan to investigate both issues in future research.

References

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Commun. ACM*, *18*, 509–517. URL: http://doi.acm.org/10.1145/361002.361007. doi:10.1145/361002.361007.

Bifet, A., Frank, E., Holmes, G., & Pfahringer, B. (2010a). Accurate ensembles for data streams: Combining restricted hoeffding trees using stacking. In M. Sugiyama, & Q. Yang (Eds.), *Proceedings of 2nd Asian Conference on Machine Learning* (pp. 225–240). Tokyo, Japan: PMLR volume 13 of *Proceedings of Machine Learning Research*. URL: http://proceedings.mlr.press/v13/bifet10a.html.

Bifet, A., Holmes, G., & Pfahringer, B. (2010b). Leveraging bagging for evolving data streams. In J. L. Balcázar, F. Bonchi, A. Gionis, & M. Sebag (Eds.), *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part I* (pp. 135–150). Berlin, Heidelberg: Springer Berlin Heidelberg. URL: http://dx.doi.org/10.1007/978-3-642-15880-3-15. doi:10.1007/978-3-642-15880-3-15.

Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., & Gavaldà, R. (2009). New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* KDD '09 (pp. 139–148). New York, NY, USA: ACM. URL: http://doi.acm.org/10.1145/1557019.1557041. doi:10.1145/1557019.1557041.

Brzeziński, D., & Stefanowski, J. (2011). Accuracy updated ensemble for data streams with concept drift. In E. Corchado, M. Kurzyński, & M. Woźniak (Eds.), *Hybrid Artificial Intelligent Systems: 6th International Conference, HAIS 2011, Wroclaw, Poland, May 23-25, 2011, Proceedings, Part II* (pp. 155–163). Berlin, Heidelberg: Springer Berlin Heidelberg. URL: http://dx.doi.org/10.1007/978-3-642-21222-2-19. doi:10.1007/978-3-642-21222-2-19.

Candanedo, L. M., & Feldheim, V. (2016). Accurate occupancy detection of an office room from light, temperature, humidity and {CO2} measurements using statistical learning models. *Energy and Buildings*, *112*, 28 – 39. URL: http://www.sciencedirect.com/science/article/pii/S0378778815304357. doi:https://doi.org/10.1016/j.enbuild.2015.11.071.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1–30. URL: http://dl.acm.org/citation.cfm?id=1248547.1248548. doi:2007-03550-001.

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* KDD '00 (pp. 71–80). New York, NY, USA: ACM. URL: http://doi.acm.org/10.1145/347090.347107. doi:10.1145/347090.347107.

Gama, J. a., Rocha, R., & Medas, P. (2003). Accurate decision trees for mining high-speed data streams. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* KDD '03 (pp. 523–528). New York, NY, USA: ACM. URL: http://doi.acm.org/10.1145/956750.956813. doi:10.1145/956750.956813.

Gomes, H. M., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfharinger, B., Holmes, G., & Abdessalem, T. (2017). Adaptive random forests for evolving data stream classification. *Machine Learning*, *106*, 1469–1495. URL: https://doi.org/10.1007/s10994-017-5642-8. doi:10.1007/s10994-017-5642-8.

Jaber, G., Cornuéjols, A., & Tarroux, P. (2013a). A new on-line learning method for coping with recurring concepts: The adacc system. In M. Lee, A. Hirose, Z.-G. Hou, & R. M. Kil (Eds.), *Neural Information Processing: 20th International Conference, ICONIP 2013, Daegu, Korea, November 3-7, 2013. Proceedings, Part II* (pp. 595–604). Berlin, Heidelberg: Springer Berlin Heidelberg. URL: http://dx.doi.org/10.1007/978-3-642-42042-9-74. doi:10.1007/978-3-642-42042-9-74.

Jaber, G., Cornuéjols, A., & Tarroux, P. (2013b). Online learning: Searching for the best forgetting strategy under concept drift. In M. Lee, A. Hirose, Z.-G. Hou, & R. M. Kil (Eds.), *Neural Information Processing: 20th International Conference, ICONIP 2013, Daegu, Korea, November 3-7, 2013. Proceedings, Part II* (pp. 400–408). Berlin, Heidelberg: Springer Berlin Heidelberg. URL: http://dx.doi.org/10.1007/978-3-642-42042-9-50. doi:10.1007/978-3-642-42042-9-50.

Kargupta, H., & Park, B. H. (2004). A fourier spectrum-based approach to represent decision trees for mining data streams in mobile environments. *IEEE Transactions on Knowledge and Data Engineering*, *16*, 216–229. doi:10.1109/TKDE.2004.1269599.

Kargupta, H., Park, B. H., & Dutta, H. (2006). Orthogonal decision trees. *Knowledge and Data Engineering, IEEE Transactions on*, *18*, 1028–1042. doi:doi.ieeecomputersociety.org/10.1109/TKDE.2006.127.

Kithulgoda, C. I., & Pears, R. (2016). Staged online learning: A new approach to classification in high speed data streams. In *2016 International Joint Conference on Neural Networks (IJCNN)* (pp. 1–8). doi:10.1109/IJCNN.2016.7727173.

Knapp, M. P. (2009). Sines and cosines of angles in arithmetic progression. In *Mathematics Magazine* (pp. 371–372). Mathematical Association of America volume 82 of *5*. doi:https://doi.org/10.4169/002557009X478436.

Lichman, M. (2013). UCI machine learning repository. URL: http://archive.ics.uci.edu/ml.

Mena-Torres, D., & Aguilar-Ruiz, J. S. (2014). A similarity-based approach for data stream classification. *Expert Systems with Applications*, *41*, 4224 – 4234. URL: http://www.sciencedirect.com/science/article/pii/S0957417413010300. doi:http://dx.doi.org/10.1016/j.eswa.2013.12.041.

Moreira-Matias, L., Gama, J., Ferreira, M., Mendes-Moreira, J., & Damas, L. (2016). Time-evolving o-d matrix estimation using high-speed gps data streams. *Expert Systems with Applications*, *44*, 275 – 288. URL: http://www.sciencedirect.com/science/article/pii/S0957417415006053. doi:https://doi.org/10.1016/j.eswa.2015.08.048.

Park, B. H. (2001). *Knowledge Discovery from Heterogeneous Data Streams Using Fourier Spectrum of Decision Trees*. Ph.D. thesis Washington State University Pullman, WA, USA.

Sakthithasan, S., Pears, R., Bifet, A., & Pfahringer, B. (2015). Use of ensembles of fourier spectra in capturing recurrent concepts in data streams. In *2015 International Joint Conference on Neural Networks (IJCNN)* (pp. 1–8). doi:10.1109/IJCNN.2015.7280583.

Sripirakas, S., & Pears, R. (2014). Mining recurrent concepts in data streams using the discrete fourier transform. In L. Bellatreche, & M. K. Mohania (Eds.), *Data Warehousing and Knowledge Discovery: 16th International Conference, DaWaK 2014, Munich, Germany, September 2-4, 2014. Proceedings* (pp. 439–451). Springer International Publishing. URL: http://dx.doi.org/10.1007/978-3-319-10160-6-39. doi:10.1007/978-3-319-10160-6-39.

Wang, H., Fan, W., Yu, P. S., & Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* KDD '03 (pp. 226–235). New York, NY, USA: ACM. URL: http://doi.acm.org/10.1145/956750.956778. doi:10.1145/956750.956778.

Zhu, X. (2010). Stream data mining repository. URL: http://www.cse.fau.edu/~xqzhu/stream.html.

Highlights

- Novel incrementally adapting Fourier Classifier is proposed.
- Strategy for efficiently computing a synopsis of data is presented.
- Novel instance schema pruning method is illustrated.
- Proposed approach outperforms existing benchmark stream classifying algorithms.