Online Pattern Recognition in Multivariate Data Streams using Unsupervised Learning

Devina Desai
ddevina1@csee.umbc.edu

Tim Oates
oates@csee.umbc.edu

Vishal Shanbhag
vshan1@csee.umbc.edu

Machine Learning Lab
University of Maryland at Baltimore County
1000 Hilltop Circle
Baltimore, MD-21250

Abstract

Extracting patterns from data streams incrementally using bounded memory and bounded time is a difficult task. Traditional metrics for similarity search such as Euclidean distance solve the problem of difference in amplitudes between static time series prior to comparison by normalizing them. However, such a technique cannot be applied to data streams since the entire data is not available at any given time. In this paper, we propose an algorithm that finds patterns in data streams in an incremental manner by constantly observing only a fixed sized window through which the data stream flows. Our algorithm employs an unsupervised approach by which it randomly samples the data streams and eventually identifies good patterns by comparing samples with one another. It uses local information to find patterns globally over the data stream. The metric used for comparison is Euclidean distance between the first derivatives of the data points of the candidate patterns. Using such a metric eliminates the problem of different scales in the data streams. The output of the algorithm is a representation of the pattern in the form of normal distributions that characterize the ranges of values that might be observered. We show that our algorithm can be used to extract interesting patterns from both univariate as well as multivariate data streams.We measure the performance of our algorithm using standard metrics like precision and recall defined according to our context.

Keywords : *t-test, Incremental, sliding window, Recall, Precision*

1 Introduction

Finding similar patterns in time series databases is important in many application domains. There are many techniques that find similarity in univariate time series using a stationary model or model parameters. Euclidean distance between the time series is a well-known similarity metric [**?**].We can implement the metric in case of incremental data streams. However, it fails to find similarities between data streams of different amplitude [**?**]. For example, the stock value of company x varies from 100 to 200 whereas that of company y varies in the range of 10 to 20, then using simple Euclidean distance, it is difficult to find similarity.

In this paper, we propose to use Euclidean distance between the first derivative of the data points in the data streams. We use a windowing technique by which there is a fixed length window of data visible to the algorithm at any time and the data enters the window at one end and leaves at the other.Therefore,

data is lost as it moves out of the observation window. At any time, we store only the information in the window and lose all previous data. The algorithm monitors the data flowing through the window and picks up sets of consecutive data points as potential patterns. We maintain a list of most similar windows of data sampled from the stream. A *t-test* is used to compare these similarity scores to an appropriate reference distribution to determine if the candidate is an instance of a pattern. The algorithm is incremental and can be applied to most of the real-time data. We assume that the data sequence is uniformly spaced in time.

2 Related Work

The problem of similarity search in *static* time series has been extensively dealt with. A static time series is defined as the one that is completely available to the algorithm at any time. There are a number of techniques describing similarity metrics for comparing either entire time series or smaller subsequences within them. [**?**] proposed to use *Euclidean distances* between the lower frequency components of the time series by performing *DFT* to convert to frequency domain. The assumption was that the high frequency components are generally noise. This approach may reduce the dimensionality but the drawback is that Euclidean distance only works for matching subsequences of equal length. Hence, [**?**] proposed to use *time-warping* distance that uses *dynamic programming* but is computationally very expensive. Yet another approach discusses aligning subsequences before matching them [**?**]. The similarity is based on the time-warping distance between the aligned subsequence. [**?**] transformed the time series into its *ARIMA* model parameters and found that the *LPC cepstrum* of these parameters could be used as an effective metric to compare the underlying time series.

Pattern recognition from *static* time series involves identifying recurring shapes in the series. Time series similarity metrics as described above are often employed in mining for these shapes. Some related work include [**?**], where probabilistic models for local patterns and global shapes are defined based on a prior distribution of deviation from a basic shape. [**?**] uses local models to predict local patterns. They survey the technique of locally weighted learning and discuss distance functions and smoothing parameters to discover these patterns. In [**?**], the problem of finding rules relating to patterns in different time series is discussed. Patterns are discovered from the time series by discretizing it and using a sliding window technique to cluster similar subsequences as interesting patterns.

The problem of analyzing streams of data points in an incremental manner is a very challenging one because of the limitations of bounded memory and bounded time. The problem of maintaining aggregates and statistics over data streams has been dealt with in [**?**]. They propose a sliding window model to compute and maintain aggregates over data streams with respect to the last N elements seen so far in an incremental manner. In [**?**], the authors have presented an incremental algorithm for clustering data points in large size databases. Their work is based on the density based clustering algorithm *DBSCAN* wherein they have proposed efficient methods for incremental insertions and deletions in the existing clustering. [**?**] have presented a constant factor approximation to the *K-median* algorithm in the data stream model of computation in a single pass. Their objective was to maintain a consistently good clustering over a stream of data points using small amount of memory and time.

Our Contribution We propose an algorithm that does not assume that data to fit in any particular model. The algorithm adopts an incremental approach. Hence the time-series need not be static. Patterns in multivariate data streams can also be discovered. The temporal information about the data is preserved, hence we can formulate rules regarding the correlation of the individual time series. For example, given two data streams describing the daily temperature and snowfall at a city, we can formulate a rule stating that when temperature falls below x^0 **C** and remains for some number of days, then the snow

depth will be around y inches for the same number of days. Our approach is scalable to n-dimensional data streams where n is relatively a large number. Hence we can predict correlations between n such data streams.

Unlike simple Euclidean distance that requires linear transformations such as amplitude shifting for data streams on different time scales [**?**], our technique does not require this type of preprocessing. This is advantageous since in non-static data we may not know the scale in advance. [**?**] uses a training set and fits training query specific models. The algorithm uses bounded memory, however it tries to fit training data only in the region of a local query. The *Landmark model* in [**?**] applies subsets of basic transformations like amplitude scaling, shifting *etc* to find minimal feature sets for the query data. The drawback of the above approach is that it requires training data to perform supervised learning. The learning in our algorithm is totally unsupervised, hence it does not require separate data or teacher. A very closely related approach similar to ours in static time series is [**?**]. It segments time series and matches similar segments. Our algorithm differs in that it preserves small deviations in the time series.

Certain parameters of the algorithm are user-defined and hence may be generic across different applications. One suitable application for our work is the speech domain. We may be interested in finding frequently uttered words. Our idea extends from [**?**] applied to speech data; it finds patterns in univariate time series by training a learner using probabilistic methods and then classifying unseen patterns with the teacher. It is a batch mode algorithm, the time series is static. Our algorithm can find patterns in multivariate data streams, is unsupervised and incremental.

Organization of Paper Section 3 contains definitions of some basic terms and their concepts. In section 4 we describe the basic idea involved in the algorithm. Section 5 talks about the implementation and the datasets on which the algorithm is tested. In section 6, we look at results of the experiments and it's evaluation based on well-defined statistical metrics. We mention future work in Section 7 and finally conclude about the algorithm in section 8.

3 Basic Definitions

The Euclidean distance between two points $[x_1, \ldots, x_n]$ and $[y_1, \ldots, y_n]$ in *n-dimensional* Euclidean space is computed as

$$\sqrt[2]{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2}$$

We propose to use Euclidean distance between the first derivative of the data points. For two time series X and Y with subsequences $[x_1, \ldots, x_n]$ and $[y_1, \ldots, y_n]$, the above metric is computed as

$$\sqrt[2]{(|x_2 - x_1| - |y_2 - y_1|)^2 + \ldots + (|x_n - x_{n-1}| - |y_n - y_{n-1}|)^2}$$

where $|x_i - x_{i-1}|$ is the slope between any two adjacent points of the data stream.

Another metric used in the discovery of patterns is the statistical-based t-test. The t-test assesses whether the means of two samples are statistically different from each other [**?**]. Given two distributions X and Y and a value for significance level α, the t-test value is given as

$$\frac{X_t - Y_t}{\sqrt{\frac{v_1}{n_1 - 1} + \frac{v_2}{n_2 - 1}}}$$

where v_1 and v_2 are variances, X_t, Y_t are the means and n_1 and n_2 are number of data points in the two distributions respectively. α value used in the t-test can be used to determine the probability of making an error in rejecting the null hypothesis that the means of two distributions are same. We use the t-test to find the difference between two distributions, pattern and it's matches and noise and it's matches. We assume that the distributions are normal. The input to the algorithm are the data points of the distributions. The means and standard deviations are estimated from them.

In our algorithm we determine the probability of the algorithm correctly predicting a pattern given that there is a pattern hidden in the data stream denoted by δ. We also define two standard metrics like recall and precision to evaluate the performance of the system. Let $|\mathbf{S}|$ be the number of patterns in the data stream and $\mathbf{A}$ be the pattern set retrieved. Note that $\mathbf{A} \cap \mathbf{S}$ is the set of relevant patterns retrieved by the algorithm. Then recall is defined as the fraction of the relevant patterns that are retrieved.

$$Recall = \frac{|A \cap S|}{|A|} \tag{1}$$

Precision is defined as the fraction of the retrieved patterns that are relevant.

$$Precision = \frac{|A \cap S|}{|S|} \tag{2}$$

Increasing significance level α increases precision but decreases the recall value and vice versa. By increasing α, the test becomes less rigorous, as a result of which some uninteresting candidate patterns will qualify as patterns.

4 Proposed Idea

We propose an algorithm that effectively finds patterns in a data stream by making a single pass over the data. We compare the results of our algorithm with a sequential algorithm that repeatedly scans the time series to find patterns.

As shown in figure 1, there is a fixed length

Figure 1: Data Stream through observation window

observation window through which the univariate data stream flows. At any time only the data present in the window is visible to the algorithm. We sample n consecutive data points at the end of the observation window at random time intervals. Here, n is equal to the desired pattern width. These n data points form the candidate patterns. Such potential patterns are stored in a **pattern bucket**. The number of patterns in the bucket k is user-defined. The data stream advances by a point and the oldest data is lost. As long as there is a pattern in the bucket, sample sets of data points of length n are picked from the observation window and matched linearly with those in the bucket. The metric used is Euclidean slope difference between the data points of the candidate pattern and the match pattern as described before. If the slope difference is less than a threshold value ϵ, then it is stored as one of the best matches for that pattern. We store p best matches for each pattern.

To compute the credibility of a candidate pattern, we need to distinguish it from noise. We use t-test to compare candidate pattern distribution with a noise distribution. For this purpose we artificially generate the noise distribution for the given data stream. At random time intervals we sample from the data stream two sets of data points, each of the size of the pattern width. Each time we compute the similarity metric between these pairs of points and record the similarity value. We construct a distribution of the values recorded in this manner, always maintaining the p best matches for noise observed this far. The mean and standard deviation of this distribution forms a representation for the noise since the sampling is completely random.

Similarly for each of the candidate pattern, we can compute its similarity value with each of its p stored best matches. The mean and standard deviation of the distribution of these values forms the corresponding representation for each of the candidate patterns. The two distributions are shown in figure 2. We define the null hypothesis and the alternate hypothesis for the t-test. The null hypothesis H_ϕ states that the noise and pattern-best matches distribution are the same.

$$H_\phi = no\, difference\, in\, distributions$$

$$H_1 = there\, is\, difference\, in\, distributions$$

According to the alternate hypothesis, the two distributions are different. α denotes the significance level (Type I error) i.e. the probability of rejecting the null hypothesis when it is true. If the distributions have means that are statistically different, then the *t-test* value is high. The t-test here is essentially an example of pattern-to-noise ratio.

If, for a given α, the *t-test* value exceeds some

Figure 2: Pattern-Best Match and Noise Distribution

user-defined valueε it is declared to be a good pattern else it is declared as useless and discarded from the pattern bucket. The representation of the promoted patterns is set of normal distributions with a mean and standard deviation for each value observed in the candidate and it's best matches. The process of extracting candidate patterns from the bucket and matching them are performed concurrently. Note that ϵ, ε, α are user-defined. The algorithm learns by itself over a period of time as it starts finding better matches to the candidate patterns in the bucket, given a large number of data points in the stream.

We measure the performance of the system by base-lining it with a sequential algorithm that exhaustively scans the data to discover patterns. The sequential algorithm requires static time series. We assume the ground truth is that the sequential algorithm generates patterns with precision and recall as 1. We use a conditional probability model to predict the output of a pattern. Let f be the frequency of the occurrence for a pattern $\pi \in \mathbf{S}$ in the data stream and r be the prior probability of the pattern π. Let $Prob(\pi \in \mathbf{A}/\pi)\, be\, \delta$
Using the definition of conditional probability,

$$\delta = \frac{Prob(\pi \in \mathbf{S}/\pi \in \mathbf{A}) * Prob(\pi \in \mathbf{A})}{Prob(\pi)}$$

The probability that the pattern will be detected depends on the α used for the t-test. If the α value is high, the probability to obtain a patten increases since the power of the test increases which indicates that the difference in the pattern and noise distribution is high. Also, if α decreases we are ready to make few errors in prediction. If the frequency of occurrence of a pattern is high, then possibility that it may be picked randomly is high. Experimentally it can be assumed that if k be a constant, then

$$\delta = \frac{k * \alpha * f}{r}$$

As a verification to the t-test, we compute the power of the test. It is given by $(1 - \beta)$, the probability of correctly predicting H_1 given it is true. If for a given α the t-test value is high, then the power of the test is greater, since there is a significant difference in the means and deviations of the two distributions, the pattern distribution and noise distribution. The lower the α the lower the power.
For the multivariate data streams, we pick data points across it as shown in figure 3 The data points of each streams are stored together as a single candidate pattern.

Figure 3: multivariate data streams through observation window

The rest of the algorithm remains the same except that now each pattern will contain data points from each of the data streams.
As mentioned, the sequential algorithm repeatedly scans the time series to form patterns. Note that consecutive patterns from the sequential algorithm are counted as a single pattern. This assumption is justified by the fact that since every potential pattern from the sequential algorithm differs only by a single point, successive patterns have a high degree of overlap in them. Hence, it is appropriate to consider them as one. Also, when we match the patterns from the sequential and incremental algorithm we allow the boundary points of the

pattern to be different. Figure 4 shows the algorithm for the sequential algorithm.

Form patterns of fixed length size
$For\ i \leftarrow 0\ to\ |timeseries| - patternwidth$
$pattern_{i,j} \leftarrow datapoint_i to datapoint_{i+patternwidth}$

Compute slope difference between adjacent points of each pattern
$For\ i \leftarrow 0\ to\ noofpatterns$
$For\ j \leftarrow 0\ to\ patternwidth$
$slopediff_i + \leftarrow |pattern_{i+1,j} - pattern_{i,j}|$

Compute sum of slope differences of each pattern with all other patterns
$For\ i \leftarrow 0\ to\ noofpatterns$
$For\ j \leftarrow 0\ to\ noofpatterns$
$slope_{i,j} + \leftarrow |slopediff_i - slopediff_j|$

Compute the confidence value for each pattern
$For\ i \leftarrow 0\ to\ noofpatterns$
$If slope_{i,j} < x\ then$
$confidence_i + \leftarrow 1$

Decide on the credibility of the patterns
$For\ i \leftarrow 0\ to\ noofpatterns$
$If\ confidence_i > y\ then$
declare it as a pattern

Figure 4: The Sequential algorithm

5 Implementation

We have implemented the algorithm on univariate as well as multivariate data streams. We intend to make a generic algorithm that will perform satisfactorily over a large number of domains. We used application domains varying from stock history data to astronomical objects, weather data and medical data. A sequential algorithm was coded and used as a benchmark for decisions on patterns. We compare the results obtained for different datasets. We use equations (1) and (2) to evaluate the performance of our algorithm.

The algorithm required multiple processes happening concurrently. Hence the system is implemented in Java which supports multi threading. Each matching and selecting process forks a thread that sleeps on a particular condition like observation window empty and wakes up on notifications like pattern bucket full. The representation of the pattern is a normal distribution with mean and very small standard deviation which indicates that it is indeed a pattern.

6 Experiments

The algorithm is tested for univariate as well as multivariate data streams. We first present the results of univariate data streams below.

Figure 5: Precision/Recall for number of data points

Figure 6: Accuracy for different pattern widths

In figure 5, we observe the precision and recall for the synthetic dataset[**?**]. This data set is designed for testing indexing schemes in time series databases. The data appears highly periodic, but never exactly repeats itself. The precision and recall values improve as the number of data points in the stream increases. The learning being unsupervised gets better as the amount of data increases as there may be more information in the data stream as it progresses. Secondly, we observe the impact of different pattern widths on the algorithm for the same dataset [**?**].

From figure 6, it is seen that the precision and recall values improve as the width of the pattern increases. For very small pattern widths, there may not be any distinguishing characteristic features to mark them as patterns. However, patterns are discovered as the number of data points increases. Hence altering the pattern width by stretching it on either sides or increasing the width leads to better results. However, we observe a fall in accuracy after

the pattern width increases beyond a certain value. There may not be patterns in the data streams of lengths greater than a maximum value.

Figure 7: Precision/Recall Values for Different Pattern Widths in ECG Data

We present the results of the experiments on ECG data [**?**] in figure 7. We observe that the peak precision and recall values are obtained for a pattern width of 8 data points after which the values begin to decrease. For the synthetic data the peak was observed for a width of 10 points. Thus, the factor of pattern width is application-specific. In figure 8, we observe

Figure 8: Precision/Recall Values for Different No of Data points in ECG Data

a similar trend in the precision and recall values except the fact that the recall remains almost constant after a fixed number of data points. Note that we observe similar behavior in the two datasets but with different values of precision and recall.

Figure 9: Precision/Recall Values for Different Pattern widths in weather Data

In figure 9, present the results for multivariate time series data [**?**]. The dataset has two time series, one describing the daily precipitation values over a period of two years and the other stream the visibility for the same days. We could expect a pattern stating that if the precipitation crosses some value x then the visibility will be reduced below some value y. The figure shows the metric values for the algorithm run on the dataset. The values are less than that of the univariate data streams. It is difficult to extract correlating patterns from multivariate data streams in an unsupervised manner incrementally because they are limited by the number of their occurences in the data streams. Hence the probability that such candidates are picked is relatively low.

Also, in each of the experiment results the recall value is higher than the precision. This illustrates the fact that our algorithm outputs fewer patterns that are not useful.

7 Future Work

The current algorithm outputs pattern of fixed size. The size of the pattern is user-defined. However, in real data, the pattern width is not known. It may be the case that the observed pattern may be a part of a larger pattern. Hence an enhancement to the algorithm would be to start with a smaller pattern window, apply the algorithm and then stretch it on either side. As we observe that the characteristics of the pattern change or there are no distinct patterns observable of that width, we may cease to expand it. We shall thus get patterns of different sizes. This is useful especially since the learning is totally unsupervised, we do not have any information about the hidden patterns in the dataset. Hence, we cannot predict the exact length of the pattern. Therefore, the above addition to the algorithm will be a good enhancement.

Since we maintain temporal information in multivariate data streams as shown in figure 3, we could predict the rules describing the correlation between each of the univariate data streams in the multivariate one. Every formed pattern will have data points from each of the data streams. We can demarcate the points for the different data streams. Based on the characteristic exhibited by each, we can formulate rules. For example, if we observe a pattern as described in the figure 3, we can formulate rule stating that if x rises then y falls. Predicting rules in an incremental manner is not been addressed exhaustively by the data mining community.

8 Conclusions

We observe that our algorithm outputs good estimate for patterns using finite memory and bounded time. The memory required for the algorithm is $O(k \cdot d \cdot p)$

where k is the number of potential patterns in the bucket p denotes the number of best matches to be stored for each potential pattern and d is the number of univariate time series. Also the computational complexity is $O(d \cdot k^2)$ per observation window through which the data flowss.

Note that the memory required is much less than of most of the time series mining algorithms which have the space complexity at least linear in N where N is the length of the time series. The computational complexity for our algorithm is at the maximum linear with respect to the length of time series since it makes only a single pass over the data. The algorithm is generic across application domains. The above results show that the algorithm performs reasonably well on different domains which could be univariate or multivariate. Hence, the approach is not data-dependent and dimension-dependent. However,we agree that there may be application specific algorithms that may perform better than our approach on the dataset for which it was designed. We claim that our approach has an equilibrium between the various factors involved in data stream mining. The advantages with respect to these factors could be summarized as follows:

(1) Precision and Recall values of about 0.5 is good for applications across different domains.

(2) The algorithm uses bounded memory and time

(3) The learning is totally unsupervised

(4) The approach is incremental.