A Project Report

On

Comparitive Analysis of Cognitive Frameworks for IoT Analytics

BY

Prasanth Yadla

2013B5A7561H

Final Report

Under the supervision of

Prof. Chittaranjan Hota

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF

CS F367: DESIGN PROJECT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI

HYDERABAD CAMPUS

(March 2017)

ACKNOWLEDGEMENT

I would like to thank Prof. Chittaranjan for his constant support.His vast experience has been valuable in driving the project through every stage.

I would also like to thank Mr. Sanket Mishra for his guidance & motivation everyday.He has been a crucial part of my project.

I would like to thank all my team members for their contributions and discussions helped me gain valuable insights.

Birla Institute of Technology and Science-Pilani, Hyderabad Campus

Certificate

This is to certify that the project report entitled "**ComparitiveAnalysis of Cognitive Frameworks for IoT Analytics**" submitted by Prasanth Yadla (ID NO. 2013B5A7561H) in partial fulfilment of requirements of the course CS F367, Design Project Course, embodies the work done by him under my supervision and guidance.

Date:
(Prof. Chittaranjan Hota)

BITS-Pilani,
Hyderabad Campus

Contents

Abstract

A recent study by McKinsey & Company estimates that the annual revenue of IoT applications may amount upto $11.1 trillion by 2025 globally.The benefits of IoT in our daily lives are countless, ranging from smart home devices (like refrigerators, air conditioners) to smart buildings and vehicles.

From data-centric perspective , the amount of information recieved by each IoT node,sensor and device can be very voluminous to analyze.There lies the emergence of a new field called IoT Analytics.The goal is to gain valuable knowledge insights from the data and possibly fire respective automation rules with authentication from the user.

In this project, we present how existing models can be combined for creating knowledge discovery.

We also present a new technique by extending a mathematically efficient approach to analyze IoT Time Series Data.

Introduction and Theory

Since the boom of Internet of Things (IoT) , sensor devices are often used to obtain new insights about our surrounding world.This created the demand for new methods to structure and represent the information.

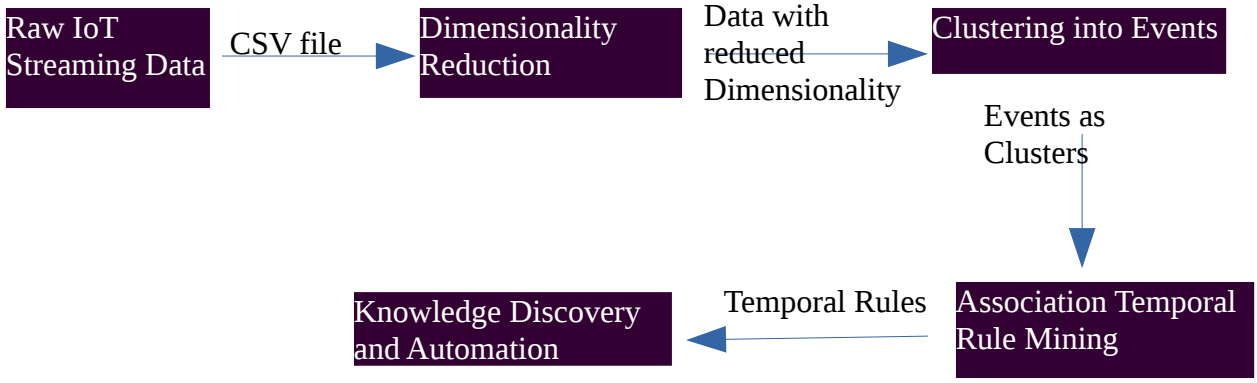
In recent years, the use of semantic concepts have been adopted since there is an abstract representation of the raw data.

In this paper, we call these abstract representations as Events.Each Event may consist of one or more instances of different sensor data at a particular time.For example , the event of weather being Sunny implies that the Temperature sensor reaches a peak of ~30 deg celc and humidity sensor outputs 40% relative humidity.While a cloudy event implies that the Temperature sensor reached ~25 deg celc and humidity sensor outputs ~65% relative humidity.

Many more sensor outputs may be clubbed together form conglomerate events.

We develop a simple Framework for analyzing IoT data and gain useful knowledge.

The following diagram illustrates the same:



At each stage, we may use one or more unique techniques for modelling and processing the correspoding data.However in this paper we use the following:

- Dimensionality Reduction: - Symbolic Aggregate Approximation (SAX)

- Clustering into Events : - K-Means Clustering Algorithm
- Rule Mining : Markov-Based Model

We have developed a modified version for efficient Rule Mining , namely Episode Rule Generation Algorithm which can be used as an alternative to Markov Model.

Do note that each stage can use different variants of the existing algorithm like Hidden Markov Model, Apriori Algorithm , K-Means++, Hierarchical Clustering .,etc and the like.

An existing Testbed was used to generate the sensor data values like Temperature, Proximity , Sound, Light . Humidity ., etc.The next section entails these details.

Each of the following sections after these covers the algorithms used for the framework in more detail.

Interfacing of Sensor Nodes and Existing IoT Testbed

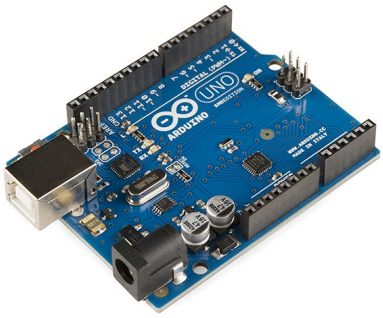


Fig 3

Fig 1

Arduino Uno Microcontroller has been used as a Node
for which sensors are to be interfaced and data is to be collected.
The data collected is then stored in the cloud by using a
ESP8266 Wifi Module.
The sketches for FC-04 Sound Sensor and DHT11Temperature, Humidity module were developed.

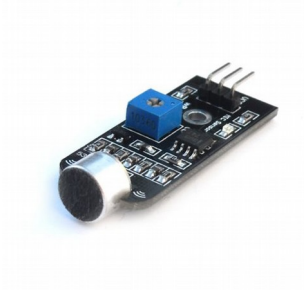


Fig 2

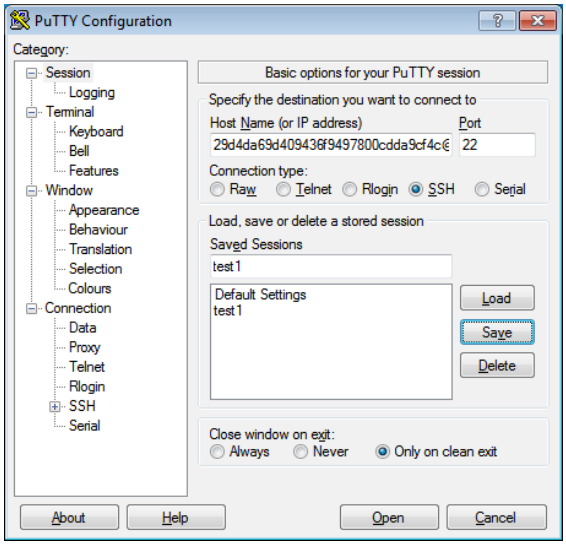
The Arduino IDE 1.0 was the platform used to write the code.Also
a breadboard with jumper wires were used to make the interfacing easier with LED lights.
PuTTy , an SSH client was being used in serial port mode to log the sensor data values into the arduino board and thereby to cloud using wifi module.An example PuTTy session configuration is shown in the figure.

A brief pseudocode of Sound Sensor Sketch is given below:

- loop() function is called everytime
- read the digital value at the PIN at which it's connected
- if the soundDetectedVal is LOW (complemented)
  then , record the timestamp and serial print it
- Otherwire loop over

A brief pseudocode of Temp&Hum sensor is given:

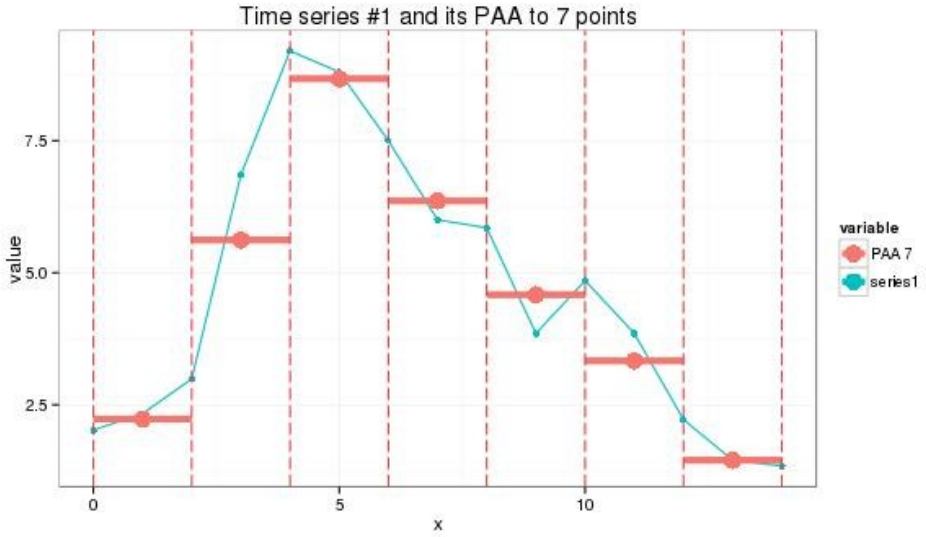
- loop over each time



- Read the corresponding temperature and humidity values from respective PINS and print

Dimensionality Reduction

SAX Algorithm



Symbolic Aggregate Approximation(SAX) algorithm is a dimensionality reduction technique used for time series data in order to transform input data to strings.

SAX can be used to introduce new metrics for measuring distance between string by extending Euclidean and PAA distances.It also uses the fact that Z-normalized time-series follow Normal Distribution

The algorithm consists of 2 steps:

1)Transforms original time-series to PAA(Piece wise Aggregate Approximation)

- A time series C of length n can be represented in a w dimensional space by a vector $\hat{C} = \hat{c}_1,\hat{c}_2,\hat{c}_3....\hat{c}_w$. The ith element of $\hat{C}$ si calculated by using the formula :

$$\hat{c}_i = (w/n) \sum c_j$$. Where j runs from $j=(n/w)(i-1)+1$ upto $j=(n/w)i$

2)Converts PAA data into strings

- A lookup table is used to match the range of floating point values to the corresponding alphabets.The mapping is user defined.

3) (Optional)Sliding Window

- In order to adapt streaming data, an internal window can be used sliding after each timestamp

Clustering to find Events

K-Means

We use dimensionally reduces SAX strings to find clusters / patterns amongst them.

Each SAX string is hashed into a particular integer using a hash function.

The hash function used is the following:

```
def Hashfunction(a):
        return ord(a)-97
```

The corresponding timestamp(location is string) is also stored along.

Therefore, they form a pair of coordinates ($hashValue_i$,$timestamp_i$) which are feeded as input to the Clustering Algorithm.

K-Means Algorithm:

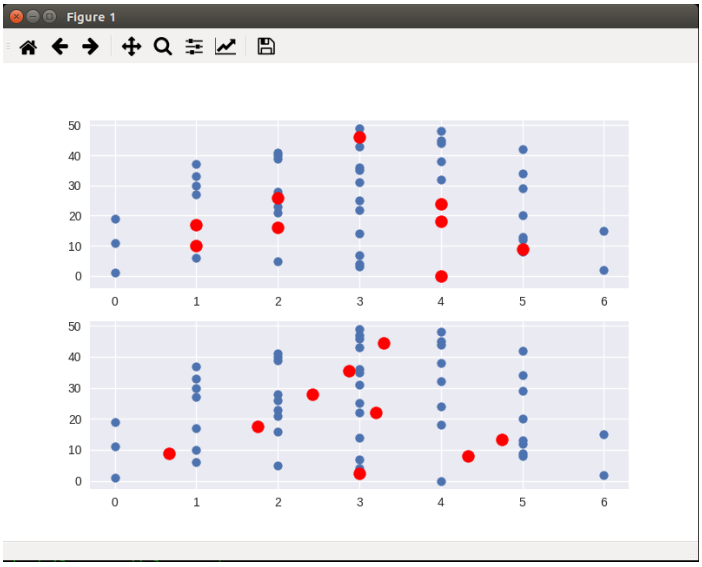
- given n observations (x_1,x_2...x_n) where each observation is a d-dimensional real vector, k-means aims to partition the n observations into k (<= n) averaged observations.
- We are to minimize:

  $$\arg\min \sum\sum ||x-v_i||^2$$

  - where, first summation runs through each point, and second summation runs through each cluster v_i

- K means finds clusters iteratively, by updating the cluster points according to the distance measure

Each clusters obtained by different sensors, when combined together with similar timestamp ,forms an event.An event consists of (Value,time) tuples for each sensor where something unique has taken place. An example clustering scenario obtained from a simple data is shown in the figure.The red points indicate the cluster points and the black dots indicate data points.In Fig 1, the upper half indicates the initial random centroids chosen.The lower half, when the algorithm converges.



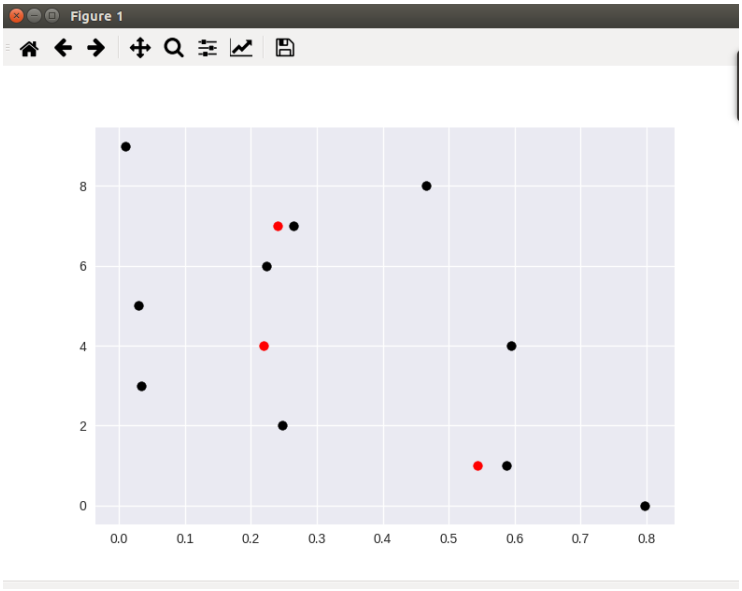


Fig 1 Fig 2

New Distance Measure used in Clustering:

Levenshtein distance:

String Metric used for measuring the difference between 2 sequences.

The Levenshtein distance between 2 words in the minimum number of insertions/deletions/substitutions required to change one word into another. All of them are single character edits.

Given below is the python snippet code of this distance.

```
# len_s and len_t are the number of characters in string s and trespectively

def LevenshteinDistance(s,len_s,t,len_t):

        # base case: empty strings
        cost = 0
        if(len_s == 0):
                return len_t
        if(len_t == 0):
                return len_s

        # test if last characters of the string match

        if(s[len_s-1]==t[len_t-1]):
                cost = 0
        else:
                cost = 1

        return min(LevenshteinDistance(s,len_s-1,t,len_t)+1,
                        LevenshteinDistance(s,len_s,t,len_t-1)+1,
                        LevenshteinDistance(s,len_s-1,t,len_t-1)+cost)


print LevenshteinDistance("kitten",6,"sitting",7)
```

The lvst distance between kitten and sitting will be outputted as 3

Temporal Rule Mining

Markov Chain Model:

Stochastic Model used model randomly changing system where future states depend only on current states

We assume the IoT Predictive Model exhibit the Markov Property

The output of Clustering Algorithm goes as input to this Model.

The implementation uses a Matrix which computes the probability of an event occuring after another event (or vice versa)

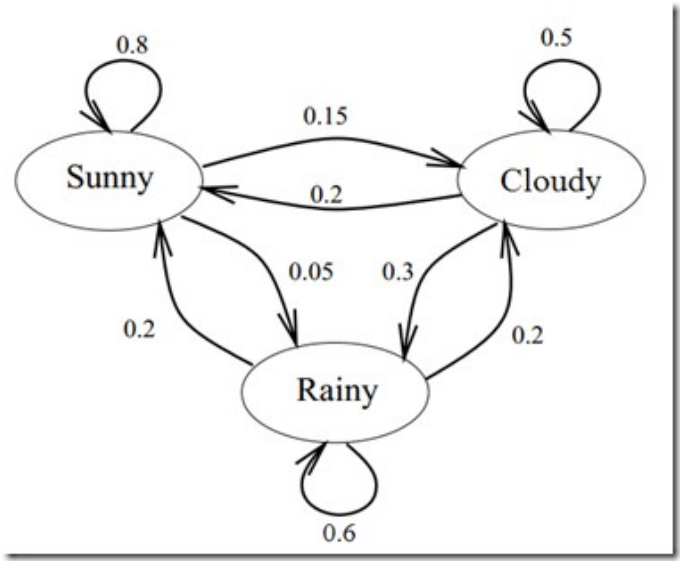
This gives us confidence and support metrics for temporal rule mining of the events

In the given figure given below, we see that the event of weather being cloudy a day after the weather is sunny has a probability of 15%

Each state transition is represented by a probability of occurence

In IoT Rule Mining, a rule is considered if it meets the threshold confidence (in here , probability)

Each state forms an event.So the output of this stage would be temporal rules between various events.



Episode Rule Mining

Very useful approach in analyzing time series data to reveal hidden patterns that are characteristic and predictive events

Traditional analysis is very inefficient and is unable to identify complex characteristics

- Episode Rule Mining uses a “bottom-up” approach, where frequent subsets of events occuring together are extended one step at a time.
- The algorithm terminates when there are no further extensions are found.
- There is ontology construction required here , unlike most traditional approaches.

Some Terminology Used Here:

- Episode :- Collection of events occuring together, mainly forming a partial ordering
- A<=B (or) A<B :- partial order, A has occured before B
- W(s,win) :- all windows in s of length win
- fr(α,s,win) = card({w Є W(s,win): α occurs in w}) / (card(W(s,win)))

The goal of the algorithm is : given a frequency threshold min_fr, window width win, discover all episodes α such that fr(α,s,win) >= min_fr

The main algorithm is given in Algo 2. Algo 1 computes the set of frequent episodes ,whose output is needed for Algo 2.

Algo 1:

Frequent Episode Generation Algorithm:

INPUT: event sequence s, win, min_fr

OUTPUT: Collection F(s,win,win_fr) of frequent episodes

1. Compute $C_1 = \{\alpha: |\alpha|=1\}$
2. l=1
3. while C_l != Ø do
4. compute $F_l = \{ \alpha Є C_l : fr(\alpha,s,win)>=min\_fr\}$
5. l = l + 1
6. compute $C_l = \{\alpha: |\alpha|=l$ and for all $\beta< \alpha$ such that $|\beta|<l$ we have $\beta Є F_{|\beta|}\}$
7. for all l do ouput F_l

Algo 2:

Episode Rule Generation Algorithm

INPUT : event sequence s, win, min_fr, confidence threshold, min_conf

OUTPUT : Episode rules that hold in s with respect to win, min_fr, min_conf

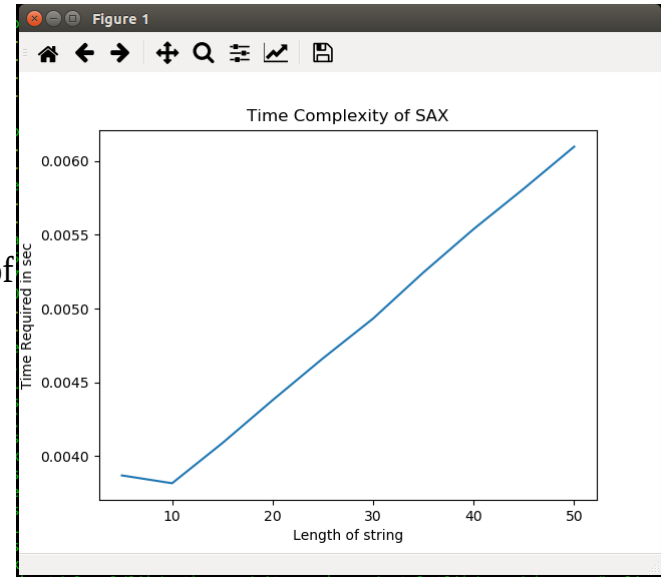
1. /*find all frequent episodes*/
2. compute F(s,win,min_fr)
3. /*generate rules*/
4. for all α Є F(s,min,min_fr) do
5. for all β< α do
6. if fr(α)/fr(β) >= min_conf then
7. output the rule β-> α and the conf. fr(α)/fr(β)

The following code snippet generates this.:-

```
31 # window size win
32 win = 3
33 j = (len(s1)/win)
34
35 def freq(alpha):
36         count = 0
37         # print alpha
38         for w in s2:
39                 i=0
40                 k = len(alpha)
41                 new_w = []
42                 while (i+k<=3):
43                         slobj = slice(i,i+k)
44                         new_w.append(w[slobj])
45                         i = i + 1
46                 count_win = 0
47                 #print new_w
48                 for q in new_w:
49                         # print q
50                         # print list(alpha)
51                         if list(alpha) == q:
52                                 #print alpha
53                                 #print q
54                                 count_win = count_win + 1
55                                 count = count + 1
56                                 #print "local window count = ", w, (count_win/3.0)
57         # print count
58         # print j
59         # return "global count = " , (count)/float(j)
60         return count/float(j)
```

Results & Improvements

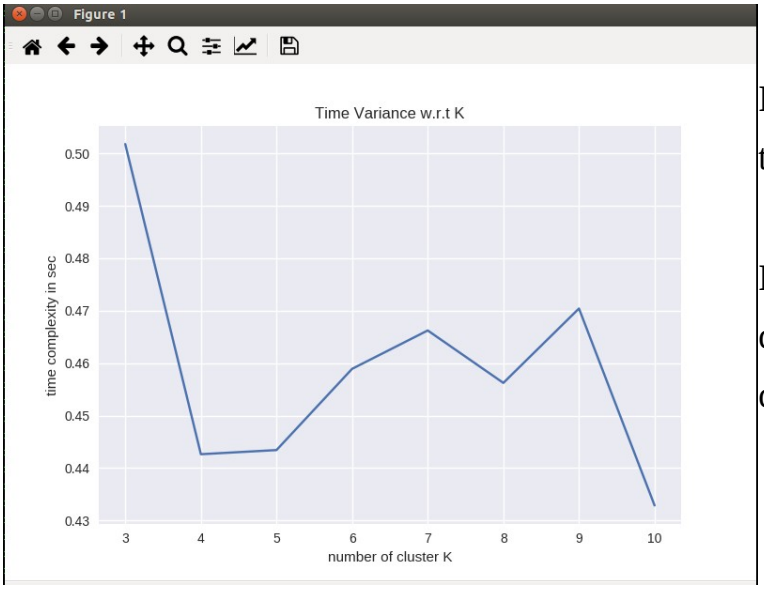
Interfacing of various sensor nodes to Arduino Board has been accomplished successfully.The sensors will be deployed into their respective locations and data will be recorded for a few days to form the dataset.The recording is due underway.



But we used a synthetic IoT dataset consisting of Temperature values to arrive at the results.

The code can easily accomodate any other dataset.

The goal to achieve knowledge discovery by generating temporal rules between events has been accomplished. Efficient Dimensionality Reduction techniques like SAX converts the raw input to strings , thereby making manipulation easier.The hashmapped strings along with corresponding timestamps are fed to clustering system and valuabe events have been generated.

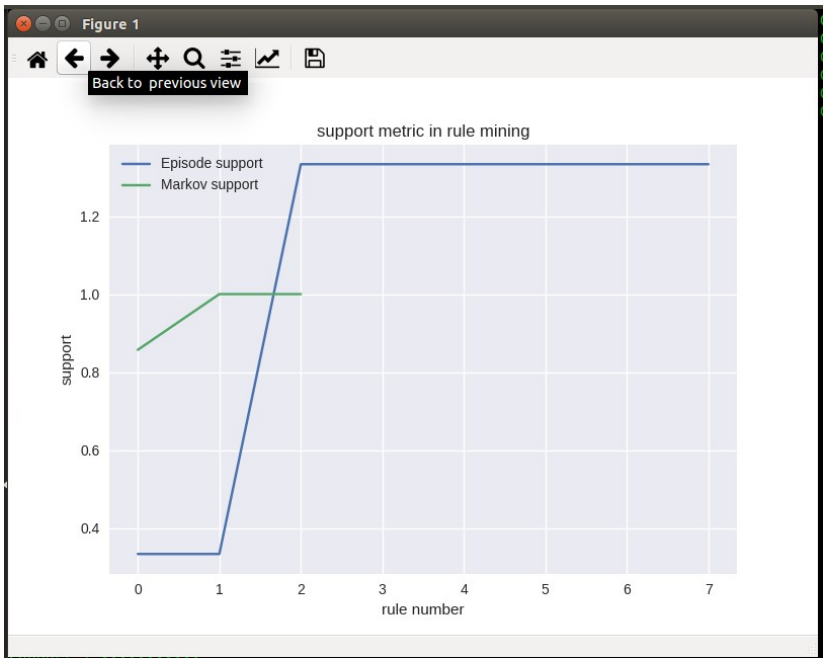


In the above figure, we plot the length of string vs the time complexity. It does increase as expected.

In the figure to the left , we plot the time complexity vs the number of clusters.Here, we conclude that a typical cluster size should be ~4/5

The events(clusters) are then fed as input to the markov chain model , which maps the alphabets to integers to encode the events.The support and confidence of all rules generated are duly noted.

The same events are once again fed into the episode rule mining algorithm , which therby produces rule whose support and confidence are noted.In Fig 3, we compare the support metrics of Markov Chain Model and Episode Rule Mining Model ., which gives interesting conclusions.



From the graph above, we can see that at as more rules are generated , the support of episode rule mining dominates over Morkov Chain Model.

Hence, episode rule mining can be used in case of scalability issues instead of Markov Model for Temporal Rule Mining Purposes.

References

- Automated Semantic Knowledge Acquisition from Sensor Data., Uni. Of Surrey
- wikipedia.org
- github.com
- Data Mining for internet of things: literature Review
- Time Series Data Mining for IOT
- Arduino.cc

APPENDIX

SAX Algorithm code:

```
#!/usr/bin/env python

import os
import numpy as np
import math
import random



class DictionarySizeIsNotSupported(Exception): pass
class StringsAreDifferentLength(Exception): pass
class OverlapSpecifiedIsNotSmallerThanWindowSize(Exception): pass




# SAX MAIN CLASS
class SAX(object):
    """
    This class is for computing common things with the Symbolic
    Aggregate approXimation method.  In short, this translates
    a series of data to a string, which can then be compared with other
    such strings using a lookup table.
    """

    def __init__(self, wordSize = 50, alphabetSize = 7, epsilon = 1e-6):

        if alphabetSize < 3 or alphabetSize > 20:
            raise DictionarySizeIsNotSupported()
        self.aOffset = ord('a')       # initial alphabet is 'a'
        self.wordSize = wordSize      # final wordsize for output
        self.alphabetSize = alphabetSize
```

```
        self.eps = epsilon
            # depending on wordsize, we need to breakup windows accordingly.
        self.breakpoints = {'3' : [-0.43, 0.43],                                    #breakpoints can be set according to ontology
                            '4' : [-0.67, 0, 0.67],
                            '5' : [-0.84, -0.25, 0.25, 0.84],
                            '6' : [-0.97, -0.43, 0, 0.43, 0.97],
                            '7' : [-1.07, -0.57, -0.18, 0.18, 0.57, 1.07],
                            '8' : [-1.15, -0.67, -0.32, 0, 0.32, 0.67, 1.15],
                            '9' : [-1.22, -0.76, -0.43, -0.14, 0.14, 0.43, 0.76, 1.22],
                            '10': [-1.28, -0.84, -0.52, -0.25, 0, 0.25, 0.52, 0.84, 1.28],
                            '11': [-1.34, -0.91, -0.6, -0.35, -0.11, 0.11, 0.35, 0.6, 0.91, 1.34],
                            '12': [-1.38, -0.97, -0.67, -0.43, -0.21, 0, 0.21, 0.43, 0.67, 0.97, 1.38],
                            '13': [-1.43, -1.02, -0.74, -0.5, -0.29, -0.1, 0.1, 0.29, 0.5, 0.74, 1.02, 1.43],
                            '14': [-1.47, -1.07, -0.79, -0.57, -0.37, -0.18, 0, 0.18, 0.37, 0.57, 0.79, 1.07, 1.47],
                            '15': [-1.5, -1.11, -0.84, -0.62, -0.43, -0.25, -0.08, 0.08, 0.25, 0.43, 0.62, 0.84, 1.11, 1.5],
                            '16': [-1.53, -1.15, -0.89, -0.67, -0.49, -0.32, -0.16, 0, 0.16, 0.32, 0.49, 0.67, 0.89, 1.15, 1.53],
                            '17': [-1.56, -1.19, -0.93, -0.72, -0.54, -0.38, -0.22, -0.07, 0.07, 0.22, 0.38, 0.54, 0.72, 0.93, 1.19, 1.56],
                            '18': [-1.59, -1.22, -0.97, -0.76, -0.59, -0.43, -0.28, -0.14, 0, 0.14, 0.28, 0.43, 0.59, 0.76, 0.97, 1.22, 1.59],
                            '19': [-1.62, -1.25, -1, -0.8, -0.63, -0.48, -0.34, -0.2, -0.07, 0.07, 0.2, 0.34, 0.48, 0.63, 0.8, 1, 1.25, 1.62],
                            '20': [-1.64, -1.28, -1.04, -0.84, -0.67, -0.52, -0.39, -0.25, -0.13, 0, 0.13, 0.25, 0.39, 0.52, 0.67, 0.84, 1.04, 1.28, 1.64]
                            }
        self.beta = self.breakpoints[str(self.alphabetSize)]
        self.build_letter_compare_dict()
        self.scalingFactor = 1
```

```
# Main driver function for PAA
    def to_letter_rep(self, x):
        """
        Function takes a series of data, x, and transforms it to a string representation
        """
        (paaX, indices) = self.to_PAA(self.normalize(x))          # paaX is just the mean of values in
each given window
        self.scalingFactor = np.sqrt((len(x) * 1.0) / (self.wordSize * 1.0))
        return (self.alphabetize(paaX), indices)
# Normalizing , Data Preprocessing Step
    def normalize(self, x):
        """
        Function will normalize an array (give it a mean of 0, and a
        standard deviation of 1) unless it's standard deviation is below
        epsilon, in which case it returns an array of zeros the length
        of the original array.
        """
        X = np.asanyarray(x)
        if X.std() < self.eps:
            return [0 for entry in X]
        return (X-X.mean())/X.std()         # mean/std_dev

# Mean finder of each sliding window
    def to_PAA(self, x):
        """
        Function performs Piecewise Aggregate Approximation on data set, reducing
        the dimension of the dataset x to w discrete levels. returns the reduced
        dimension data set, as well as the indices corresponding to the original
        data for each reduced dimension
        """
        n = len(x)                              # length of given data
        stepFloat = n/float(self.wordSize)    # number of windows
        step = int(math.ceil(stepFloat))    # rounded integer value
```

```
        frameStart = 0     # initial Frame Number / Window Number
        approximation = []      # Final value to be returned, ie., the mean
        indices = []      # indices of start and end of each window
        i = 0
        while frameStart <= n-step:
            thisFrame = np.array(x[frameStart:int(frameStart + step)])       # array containing all values
of current window/frame
            approximation.append(np.mean(thisFrame))                          # calculating mean
            indices.append((frameStart, int(frameStart + step)))            # indices updations
            i += 1
            frameStart = int(i*stepFloat)                                  # Incrementing Next Frame
        return (np.array(approximation), indices)
# Converts output of PAA function to string based on existing ontology
    def alphabetize(self,paaX):
        """
        Converts the Piecewise Aggregate Approximation of x to a series of letters.
        """
        alphabetizedX = ''    # string value
        for i in range(0, len(paaX)):   # checking for correct range in breakpoints
            letterFound = False
            for j in range(0, len(self.beta)):
                if paaX[i] < self.beta[j]:
                    alphabetizedX += chr(self.aOffset + j)   # character 'a'+offset(j)
                    letterFound = True
                    break
            if not letterFound:
                alphabetizedX += chr(self.aOffset + len(self.beta))    # otherwise go to the end
        return alphabetizedX



# The below functions are not required as of now.They are used for comparision purposes only
# We can change them later accordingly
```

```
def compare_strings(self, sA, sB):
    """
    Compares two strings based on individual letter distance
    Requires that both strings are the same length
    """
    if len(sA) != len(sB):
        raise StringsAreDifferentLength()
    list_letters_a = [x for x in sA]
    list_letters_b = [x for x in sB]
    mindist = 0.0
    for i in range(0, len(list_letters_a)):
        mindist += self.compare_letters(list_letters_a[i], list_letters_b[i])**2
    mindist = self.scalingFactor* np.sqrt(mindist)
    return mindist

def compare_letters(self, la, lb):
    """
    Compare two letters based on letter distance return distance between
    """
    return self.compareDict[la+lb]

def build_letter_compare_dict(self):
    """
    Builds up the lookup table to determine numeric distance between two letters
    given an alphabet size.  Entries for both 'ab' and 'ba' will be created
    and will have identical values.
    """

    number_rep = range(0,self.alphabetSize)
    letters = [chr(x + self.aOffset) for x in number_rep]
    self.compareDict = {}
    for i in range(0, len(letters)):
        for j in range(0, len(letters)):
```

```
            if np.abs(number_rep[i]-number_rep[j]) <=1:
                self.compareDict[letters[i]+letters[j]] = 0
            else:
                high_num = np.max([number_rep[i], number_rep[j]])-1
                low_num = np.min([number_rep[i], number_rep[j]])
                self.compareDict[letters[i]+letters[j]] = self.beta[high_num] - self.beta[low_num]

def sliding_window(self, x, numSubsequences = None, overlappingFraction = None):
    if not numSubsequences:
        numSubsequences = 20
    self.windowSize = int(len(x)/numSubsequences)
    if not overlappingFraction:
        overlappingFraction = 0.9
    overlap = self.windowSize*overlappingFraction
    moveSize = int(self.windowSize - overlap)
    if moveSize < 1:
        raise OverlapSpecifiedIsNotSmallerThanWindowSize()
    ptr = 0
    n = len(x)
    windowIndices = []
    stringRep = []
    while ptr < n-self.windowSize+1:
        thisSubRange = x[ptr:ptr+self.windowSize]
        (thisStringRep,indices) = self.to_letter_rep(thisSubRange)
        stringRep.append(thisStringRep)
        windowIndices.append((ptr, ptr+self.windowSize))
        ptr += moveSize
    return (stringRep,windowIndices)

def batch_compare(self, xStrings, refString):
    return [self.compare_strings(x, refString) for x in xStrings]

def set_scaling_factor(self, scalingFactor):
```

```
        self.scalingFactor = scalingFactor

    def set_window_size(self, windowSize):
        self.windowSize = windowSize



# Main Function Begins here


Obj = SAX()                                                    # Create Object for SAX Usage

ar = []
for x in range(100):
            ar.append(random.randrange(1,100))

            #ar = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,6,6,6,6,10,100]                    # Any Given Array Can
Be Inputted here

#print ar

print Obj.to_letter_rep(ar)                                    # Returns 1) A String representing
PAA representation of given Data
                                                               #         2) List of tuples indicating
start and end point of window
```

K-Means Algorithm code:

```
import matplotlib.pyplot as plt          # plotting library matplotlib
import numpy as np                     # statistics library
import seaborn as sns;sns.set()          # graphics and visualization library
import random as r
import math
```

```
#from JSAnimation import IPython_display
#from matplotlib import animation

# define points here.....

# Uncomment the below for randomized points
'''
points = np.vstack(((np.random.randn(150,2)*0.75 + np.array([1,0])),
        (np.random.randn(50,2)*0.25+np.array([-0.5,0.5])),
        (np.random.randn(50,2)*0.5 + np.array([-0.5,-0.5]))))

temp_points = np.random.randn(5,2)
'''
#points = [[1,0],[2,1],[3,2],[5,3]]



'''
for SAX INPUT USE THE FOLLOWING CODE:

# input s the string
# output points containing decimal equivalent

points = []
for i in s:
        points.append([])
        points[i].append(ord(i)-97)

for i in range(10):
        points[i].append(i)
```

```
'''



points = []
for i in range(10):
        points.append([])
        points[i].append(r.random())

for i in range(10):
        points[i].append(i)

# Points to be defined here:
#points = np.vstack([[1,1],[2,2],[3,3],[2.5,2.5]])      # convert to Numpy array stack to make processing easier

#print temp_points[0][0]
#print temp_points

# print points

'''
plt.scatter(points[:,0],points[:,1])
ax = plt.gca()
ax.add_artist(plt.Circle(np.array([1, 0]), 0.75/2, fill=False, lw=3))
ax.add_artist(plt.Circle(np.array([-0.5, 0.5]), 0.25/2, fill=False, lw=3))
ax.add_artist(plt.Circle(np.array([-0.5, -0.5]), 0.5/2, fill=False, lw=3))

# plt.show()
'''
```

```
# function to initialize random centroids
def initialize_centroids(p,k):
        # returns k centroids from the initial points
        #centroids = p
        #np.random.shuffle(centroids)
        #r.shuffle(centroids)
        #return centroids[:k]
        return r.sample(p,k)

# print initialize_centroids(points,3)


def eucl_dist(x,y):
        #       print x[1]
        #print y[1]
        d1 = pow(x[0]-y[0],2)
        d2 = pow(x[1]-y[1],2)
        return math.sqrt(abs(d1-d2))




# function
def closest_centroid(points,centroids):
        # returns an array containing the index to the nearest
        # centroid for each point
        '''
        distances = np.sqrt(((points-centroids[:][np.newaxis])**2).sum(axis=2))
        return np.argmin(distances,axis=0)
        '''
        # print centroids
        min_centr_points = []
        for i in points:
```

```
            min_dist=1e9

            for j in centroids:
                if eucl_dist(i,j)<min_dist:
                    min_dist = eucl_dist(i,j)
                    min_cnetr = j
            min_centr_points.append(min_cnetr)

        return min_centr_points




def move_centroids(points,closest,centroids):

        #return np.array([points[closest==k].mean(axis=0) for k in range(centroids.shape[0])])
        '''
        points = np.array(points)
        closest = np.array(closest)
        centroids = np.array(centroids)

        return np.array([points[closest==k].mean for k in range(centroids)])
        '''
        new_centroids = []
        for k in centroids:
            # print k
            index_list = []
            index_list = [i for i,x in enumerate(closest) if x==k]
            # print index_list
            s = [0.0,0]
            for x in index_list:
                s[0] = s[0] + points[x][0]
```

```
                    s[1] = s[1] + points[x][1]

            # print s
            # print new_centroids
            new_centroids.append([s[0]/len(index_list),s[1]/len(index_list)])

    return new_centroids


#plt.scatter(points[:,0],points[:,1])
#c = initialize_centroids(points,3)
#plt.scatter(centroids[:,0],centroids[:,1],c='r',s=100)

#plt.show()


#print closest_centroid(points,c)

#plt.subplot(2,1,1)
#plt.scatter(points[:][0],points[:][1])
centroids = initialize_centroids(points,3)
#print centroids

#print centroids
#plt.scatter(centroids[:][0],centroids[:][1],c='r',s=100)
#plt.show()
# c_extended = c[: , np.newaxis, :]
#print c_extended



#plt.subplot(2,1,2)
#plt.scatter(points[:,0],points[:,1])
```

```
#plt.scatter(points[:][0],points[:][1])

closest = closest_centroid(points,centroids)
#print points
#print closest

#print "points"
#print points
#print "closest centroids:"
#print closest
i = 10
while(i):
        centroids = move_centroids(points,closest,centroids)
        closest = closest_centroid(points,centroids)
#       print centroids
        i = i-1
#plt.scatter(centroids[:][0],centroids[:][1],c='r',s=100)



#plt.show()

print points


x = [p[0] for p in points]
#print x

y = [p[1] for p in points]
#print y

plt.scatter(x,y,label='plot',color='k')
#plt.show()
```

```
x1 = [p[0] for p in centroids]
y1 =  [p[1] for p in centroids]

plt.scatter(x1,y1,label='plot',color='r')

plt.show()
print centroids




'''
fig = plt.figure()
ax = plt.axes(xlim=(-4,4),ylim=(-4,4))
centroids = initialize_centroids(points,3)

def init():
        return

def animate(i):
        global centroids
        closest = closest_centroid(points,centroids)
        centroids = move_centroids(points,closest,centroids)
        ax.cla()
        ax.scatter(points[:,0],points[:,1],c=closest)
        ax.scatter(centroids[:,0],centroids[:,1],c='r',s=100)
        return
```

```
animation.FuncAnimation(fig,animate,init_func=init,frames=10,interval=200,blit=True)

'''
```