Autonomous Dynamic Learning Apprentice System

Mandar Kakade, Sourav Khemka, Parth Verma, Anshuman Chakravarty, Rahul Das

Abstract

In this paper we present a *Learning Apprentice System* (LAS) that automates the process of launching applications which are run concurrently by a user. Our system tries to optimize the set of operations a user would need to perform in order to launch various applications in coherence with each other. A simple implementation using Self Organizing Feature Maps has been constructed that demonstrates the feasibility of building such a system.

Keywords: *Unsupervised Learning, Clustering, Learning Apprentice System, Self Organising Maps*

Introduction

In this fast-paced world the way in which users work on a desktop has changed drastically. People have started exploiting the attribute of multitasking in PCs to the fullest. Thus there are a lot of applications which are to be launched in sync with each other but to do so the user has to explicitly start each and every application which can be frustrating at times. In order to curtail the user involvement in the above mentioned process we have a developed a Learning Apprentice System- ADLAS.

Learning Apprentice Systems are interactive knowledge-based consulting systems that directly confront the knowledge-acquisition bottleneck by learning from the users. ADLAS is a system which would assimilate the recent activities on a workstation and use it to simplify the tasks the user needs to perform in order to launch multiple applications.

Self Organizing Feature Maps

A self-organizing feature map (SOFM) is a type of an artificial neural network (ANN) that is trained using unsupervised learning to produce a low-dimensional, discretized representation of the input space of the training samples, called a map, and is therefore a method to do dimensionality reduction. Self-organizing maps differ from other artificial neural networks as they apply competitive learning as opposed to error-correction learning. When the number of SOM units is large, to facilitate quantitative analysis of the map and the data, similar units need to be grouped, i.e., clustered. This method is very effective because the system is able to automatically organize the applications into meaningful clusters according to their parameters.Using the SOFM to cluster documents is advantageous because of its ability to solve large and computationally expensive classification problems efficiently.

Methodology

The whole process from the activation of ADLAS to launching of fitting/relevant apps can be divided into three stages : Data Collection,Clustering,Launching.

Data Collection

Gathering relevant data of a user by tracking his behavior for a fixed amount of time. This is achieved by running a powershell script, every second, in the background. The output of this script gives us a file for every second, each file containing a list of all the processes running on the machine at that instant in csv format. Once these files are generated, we clean them by keeping those applications which have more utility for a user and were popular, in general, for specific tasks. Data Cleaning is done with the help of a python script whose output is also written in a csv file.

Next, we use these files to finally generate a single csv file containing the start and end times(Figure 1) of all the applications launched during the time frame of user monitoring. The code to accomplish this task is written in C++.

	A	B	C
1	Name	Start t	End t
2	chrome	12	70
3	sublime_text	26	75
4	DCPlusPlus	83	201
5	vlc	93	188
6	chrome	211	326
7	sublime_text	213	331
8	chrome	438	524
9	sublime_text	439	527
10	wordpad	574	713
11	mspaint	580	707
12	AcroRd32	737	834
13	WINWORD	759	823
14	chrome	930	946
15	sublime_text	932	948
16	sublime_text	977	1307
17	chrome	980	1290

Figure 1: Input Files Given to train the model

Clustering

We use *Self Organizing Feature Maps* to cluster the applications.The *Start Time* and *Running Time* of each applications are used as features to generate the clusters. SOFM is implemented using the *MATLAB nctoolbox*. We store the frequency of clusters having the same set of applications. A high frequency of a particular cluster validates its existence. We set a threshold frequency and render a cluster invalid if the frequency of the cluster is less than the threshold frequency. Finally we store the valid clusters in a text file.

	A	B	C	D	E
1					**Frequency**
2	sublime_text	chrome			9
3	wordpad	mspaint			2
4	WINWORD	AcroRd32			1
5	sublime_text	chrome	AcroRd32		1
6	vlc	DCPlusPlus			1
7	wordpad	sublime_text	mspaint	chrome	1
8					
9					

Figure 2: Clusters formed and its frequency

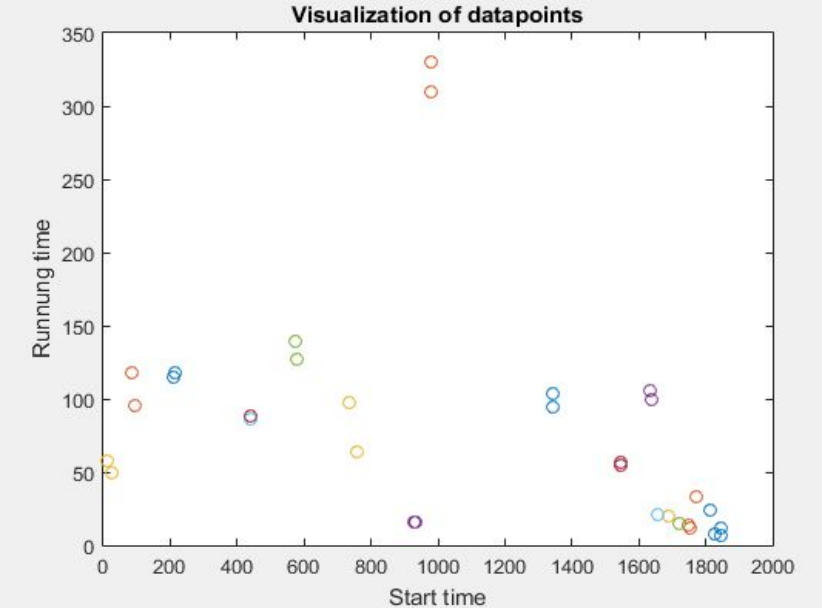

Figure 3: Visualization of clusters

Launching

When an intention of a user is detected, a c++ program compares it with the clusters generated. After a match is found, all the applications in that cluster are launched using a combination of python code and cmd command.

Conclusion

To train ADLAS we adopt a clustering technique which generates distinct clusters each representing a set of applications which are launched simultaneously by a user. Henceforth if the user intends to launch an application, ADLAS would search for the cluster to which this program belongs and thus, launch the whole set of applications in that particular cluster. Thus we see that by using ALDAS, we are able to reduce user involvement in the computer. Once a user opens an application, the entire cluster is opened. Thus we optimize time and involvement of the user which is the need of the hour.