Issues in Recording Benchmark Sensor Data

Kristof Van Laerhoven and Hans-Werner Gellersen

Department of Computing, Lancaster University,
LA1 4YR Lancaster, United Kingdom
{kristof, hwg}@comp.lancs.ac.uk

Abstract. Sensors are rapidly following computing devices in popularity and widespread use; and as a result, protocols to interface, record and process sensor data have cropped up anywhere. This position paper lists some of the 'lessons learned' in the creation and application of sets of embedded sensor data, specifically used as tools in building context aware services where sensor values get classified into context descriptions.

1 Introduction

Recorded sensor data is one of the most common tools in machine learning, and the majority of research uses it to get an idea on which algorithms might be suitable for building a system that processes and classifies the sensor data. The recordings help to get a feel on how the application's sensor signals are likely to look, and what effect a certain event has on the data (and how this can be exploited by the classification algorithms that translate this data into high-level descriptions). Acquiring a collection of datasets under characteristic circumstances is typically the first task to be undertaken after the hardware system has been built, and prior to the design of the classification algorithms.

A second typical use for recorded datasets is the evaluation afterwards: to prove that a system works, experiments need to be reproducible; to prove that algorithm A is superior to algorithm B in some respect, the first requirement is that a common 'benchmark' dataset must be used. Such evaluation datasets exist for a variety of applications and sensors, although the majority of them cover data that have been sampled at a high rate and focuses on one source, such as audio, video, ECG and EEG monitoring systems. Signals from a large and mixed variety of sensors are harder to find.

This paper will concentrate on these two uses for recorded sensor data, with a special interest in those of systems with many sensors, individually having a relatively low update rate. With this in mind, the following section will discuss some issues in the use of stored sensor data while section three will focus on a proposed toolkit that handles communication visualization, and processing of sensor data with a strong emphasis on the use of datasets.

2 The Limitations of a Dataset

This section will focus on the usage of recorded low-level sensor data for the design of classification systems only, although many of the observations that will be made in this section most likely apply for a wider range of dataset usage. Unlike traditional work on the correct, rigorous use of datasets for classification [3], it also intends to offer newer insights.

2.1 Recording Annotations

When information from sensors gets stored on disk, annotations or meta-data that is valuable for classification needs to be included to make these recorded files usable in the future. It is a common misconception to expect that the data from *any* sensor will afterwards 'speak for itself' and that adding tags to the data straight away is not necessary. This becomes more prominent in approaches that employ many sensors that individually give only a very limited view on what is happening.

2.2 Recording the Same Data

After having chosen a set of sensors to work with, datasets get often recorded in the initial process to study how well the sensors perform as a sanity check for the sensor hardware. Similar datasets are often recorded a second time, with a scenario or a fixed set of events or contexts in mind that provide the target classes. Finally, usually to provide a benchmark to the community, another dataset is recorded that has the right annotations embedded so that third parties can test their algorithms on the same datasets and compare performances. This might point to a need for generic utilities to (1) check the behaviour of sensors under certain events and (2) consistently generate datasets.

2.3 Bias Toward Offline Train-Test Algorithms

The process of recording a dataset, analyzing it, and possibly repeating these two steps ad infinitum, aligns easily with the classic approaches of training and testing a model in machine learning. This does often result, however, in the implementation of algorithms that first get trained on offline datasets, and then are tested in real-time afterwards.

For this reason, incremental machine learning algorithms tend to be neglected as their development is not that compatible with recorded datasets. This is unfortunate, since it forces the user to play a minor role (or take the user out of the loop completely) when the classification algorithm is trained. Incremental algorithms gradually adjust their internal models until their classification is found adequate by the user, who also can intervene for particular classes; this very close interaction between the system and the user in the training process is often very efficient: only poorly classified data is re-trained, and training stops when the user finds the algorithm's performance to be accurate enough.

One way to correct this bias could be the integration of *user-annotations* in the dataset, which cue the system to train for that specific data. This still requires 'universal' annotations that can be used to evaluate the classifier's performance afterwards for all of the data (when no user-annotation is present). These two layers of annotations would complicate a dataset's format a bit more.

2.4 Conclusion

In summary, most of these observations point out a need for:

- a set of tools that facilitate the recording, communication, and visualisation of real-time sensor signals in a consistent way, without requiring extra effort in the design process of the system
- a standard format (or a set of standard formats) for sensor datasets, so that logged experiments can easily be replicated on other sites, and so that identical data can be used to compare approaches

The next section will introduce work in progress to respond in part to these needs. As it essentially is a toolkit developed as an open source platform between researchers, it might serve as a mutual record for sensor processing related algorithms.

3 The CommonSense ToolKit

The CommonSense ToolKit [1] was developed out of a need for modular software components that assist in the communication, abstraction and visualisation of sensor data. CSTK's main qualities are its real-time facilities and embedded systems-friendly implementation, providing ready-to-use modules for the prototyping and construction of sensor-based applications.

It is being developed within the CommonSense [2] project, which aims to investigate the integration of multiple diverse sensors for user-level context acquisition in wearable and ubiquitous computing. The context acquisition method can be thought of as an abstract sensor for detecting context within a set of user situations for which the sensor is configured and trained.

CSTK can be used for offline analysis (i.e., using recorded data files), but is envisioned as a tool that can be applied in an online fashion (i.e., using the live data that comes streaming in). Various sensor hardware platforms are supported, including most serial protocols and embedded sensors (such as the iPAQ's internal sensors).

CSTK has three layers in which users can start using CSTK and make modifications, depending on their needs in performance, integration and complexity:

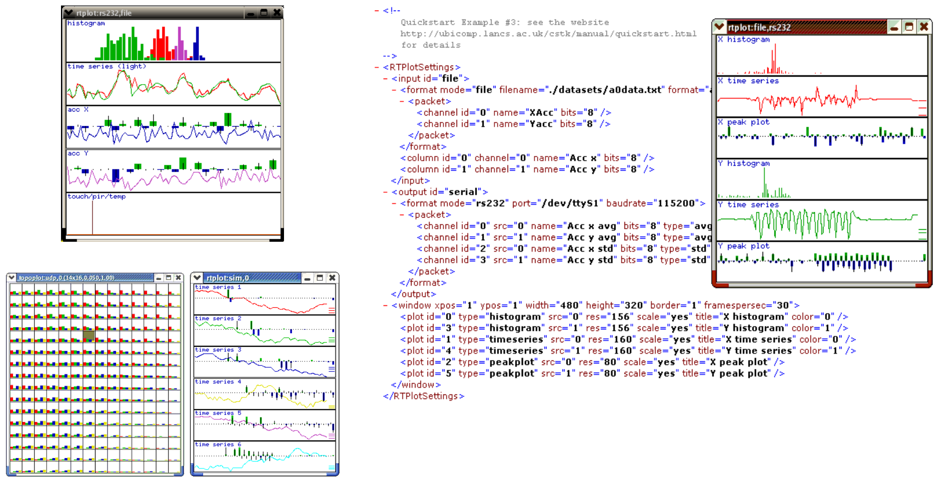
- The core source code of CSTK is written in C++ and with small low-power embedded architectures in mind (such as the iPAQ's arm processors).

- A collection of command line driven utilities that visualize, cluster and classify sensor data form a second layer. They allow real-time processing such as extracting basic features, calculating statistics and signal peak descriptors, or building topographic maps of the sensor data.

- An integrated graphical environment is planned to merge the different tools, which launches the modules, and tweaks the parameters. This is intended as the easiest and quickest way to get a grip on sensor data, but might be somewhat limited for specific applications.

CSTK runs on most operating systems that support X11 or XFree86™; this mainly includes UNIX® and UNIX-like operating systems such as Linux, BSD, Sun Solaris x86, Mac OS X™ (via Darwin), but also other platforms like OS/2 and MS Windows™ (using Cygwin), and especially distributions for handheld platforms such as Familiar [4]. It is also a GPL-licensed open source project that is freely available for download.

Figure 1 shows some examples of visualisation (updated in real-time) of streaming sensor data, and how the tools' numerous parameter values can be managed in one xml settings document. The tools are meant to be able to run on environments that have a restricted amount of resources.



```
- <!--
    Quickstart Example #3: see the website
    http://ubicomp.lancs.ac.uk/cstk/manual/quickstart.html
    for details
  -->
- <RTPlotSettings>
  - <input id="file">
    - <format mode="file" filename="./datasets/a0data.txt" format="
      - <packet>
          <channel id="0" name="XAcc" bits="8" />
          <channel id="1" name="Yacc" bits="8" />
        </packet>
      </format>
      <column id="0" channel="0" name="Acc x" bits="8" />
      <column id="1" channel="1" name="Acc y" bits="8" />
    </input>
  - <output id="serial">
    - <format mode="rs232" port="/dev/ttyS1" baudrate="115200">
      - <packet>
          <channel id="0" src="0" name="Acc x avg" bits="8" type="avg
          <channel id="1" src="1" name="Acc y avg" bits="8" type="avg
          <channel id="2" src="0" name="Acc x std" bits="8" type="std"
          <channel id="3" src="1" name="Acc y std" bits="8" type="std"
        </packet>
      </format>
    </output>
  - <window xpos="1" ypos="1" width="480" height="320" border="1" framespersec="30">
      <plot id="0" type="histogram" src="0" res="156" scale="yes" title="X histogram" color="0" />
      <plot id="3" type="histogram" src="1" res="156" scale="yes" title="Y histogram" color="1" />
      <plot id="1" type="timeseries" src="0" res="160" scale="yes" title="X time series" color="0" />
      <plot id="4" type="timeseries" src="1" res="160" scale="yes" title="Y time series" color="1" />
      <plot id="2" type="peakplot" src="0" res="80" scale="yes" title="X peak plot" />
      <plot id="5" type="peakplot" src="1" res="80" scale="yes" title="Y peak plot" />
    </window>
  </RTPlotSettings>
```

Fig. 1. Some CSTK screenshots of some of the visual tools (left), and the layout of an XML Settings Document with its corresponding tool (right).

Acknowledgements

This toolkit is built as part of the CommonSense project at the University of Lancaster (UK), and sponsored by the Engineering and Physical Sciences Research Council (EPSRC). We thank both institutions for their support, as well as SourceForge for providing additional web and code / documentation versioning support, and the other developers for extending the toolkit.

References

1. The CommonSense ToolKit (CSTK) online manual: **http://ubicomp.lancs.ac.uk/cstk/manual/** or **http://cstk.sourceforge.net/manual/**
2. CommonSense: **http://ubicomp.lancs.ac.uk/commonsense** (verified 02/04/2004)
3. Steven L. Salzberg “On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach”. In Data Mining and Knowledge Discovery, 1, 317-327 (1997), Kluwer, Boston.
4. Familiar Linux Distribution, Handhelds.org: **http://familiar.handhelds.org/** (verified 02/04/2004)