Approximate Views and compression

February 26, 2014

1 Motivation

We consider situations where we have a sensor network that is distributed over a large area. Our goal is to perform continuous analysis of the data generated by the sensors. The goal of the analysis is to detect abnormal behaviour and to create predictive models of the behaviour of the physical system (PS) that is monitored by the sensors.

We assume the following generic architecture, consisting of sensor nodes and compute nodes. Sensor nodes consist of some sensors a general purpose computer and local storatge. Compute nodes consist only of computers, potentially stronger than the computers in the sensor nodes. The compute nodes communicate with each other and with the sensor nodes. The compute nodes aggregate information from many sensors in order to estimate the global state of the PS, and to create predictive models for the dynamics of that PS. The main constrained resource is the communication bandwidth between the nodes.

As an example consider an ad-hoc sensor network consisting of smart-phones. The smart-phones communicate with a network of computers through cell phone connections and the internet. The amount of data generated by a sensor such as a video camera easily exhausts the bandwidth cellular communication network. Even when the bandwidth is not exhousted it is usually the most expensive part of operating the system, both in terms of data-communication costs and in terms of battery life.

It is therefor very desirable to design a system which operates in such a way as to minimize the amount of communication between the sensors and the compute nodes. A common approach is to use *lossless compression*. This is a good solution when possible, but it rarely decreases the communication volume by a factor bigger than 4. Our goal is to design method that will decrease the communication volumes by a factor of ten or more.

To achieve such rates we need to look towards *lossy compression* methods. When data is compressed and decompressed using a lossy compression method, the result is a *distorted* version of the original data. We say that a compression method is good if a small data *rate* (i.e. the bandwidth required to carry the compressed data) is enough to achieve low expected *distortion*. The foundational theory of lossy compression is Shannon's Rate-Distortion theory, which characterizes the achievable rate-distortion pairs.

The way in which distortion is measured is sometimes of critical importance. For example the use of *perceptual coding* in MP3 reduces the number of bit devoted to encoding frequencies to which the human ear is less sensitive.

The basic idea behind this work is that the data coming from sensors about a PS can usually be seen as a sum of two parts: signal and noise. Usually written as:

$$f(t) = s(t) + \sigma(t)w(t)$$

where $s(t)$ is the signal as a function of time $w(t)$ is white noise and $\sigma(t)$ is the amplitude of the noise. White noise is not compressable, in other words, no method of coding can decrease the bandwidth required to send it. Happily, white noise also carries no useful information about the PS, only $s(t)$ and $\sigma(t)$ carry useful information. In addition, $s(t)$ and $\sigma(t)$ are usually highly compressible. This notion can be formalized in a general way using Kolmogorov Sufficient Statistics.

Our plan is therefor to partition the signal from a sensor (or a set of sensors) into the useless and uncompressible noise part and the useful and compressible signal and noise-amplitude part.

2 Approximate Views as an abstract data type

Here is an attempt to formalize the relationship between a physical system which is continuous in time and space, the discrete sensor data extracted from it, fitting a model to this data, compressing and decompressing.

1. A *coordinate system* is the d dimensional Euclidean space R^d with agreed upon names for the coordinates. Time is always a coordinate. Other possible coordinates define the *state* which includes quantities such as location, temperature, angle, speed etc.

2. A *trajectory* is a mapping from (real) time to (real) state.

3. Let $\mathcal{M}$ be the *Model Class*. The model class defines the set of allowable trajectories. In other words, it defines what the physical system can do.

4. Let X be the raw data. The raw data is a table. The coordinates are columns with continuous values.

5. $Y = V(X)$ is an *approximate view* or a *model* of X. The model defines a trajectory in the model class.

6. $l(X), l(Y)$ are the respective sizes of X and Y in bits.

7. A query to a model has the form $A = Q(V(X), \vec{p})$ where $\vec{p}$ is a coordinate vector.

8. The distortion of a model is measured by making queries to $V(X)$ the reconstruct the rows of X. The distortion is the distance between the X and it's reconstruction. (When sensors are noisy, the reconstruction might actually be *better* than the raw data. I have yet to figure out how to quantify that).

Example: Suppose the raw data is the GPS data from a cellphone together with some measurements such as pollution level. A model of the data can be an ARMA model with a stream of correction vectors. This arma model represents a smoothed and compressed

version of the raw data. A query can be of the form: "Where was person A at time t?" Note that the particular time might not exist in the raw data. The ARMA model generates an answer using interpolation.

The approximate view serves three purposes: Compression, noise reduction and interpolation (providing values at times where no values were measured).

3 Kolmogorov Sufficient Statistic

We suppose that all of the sensors share a synchronized clock t. We think of the data collected from sensor i as a data stream:

$$s^i(\tau_1, \tau_2) = [(t_1^i, x_1^i), \ldots, (t_n^i, x_n^i)] \text{ where } \tau_1 \leq t_1^i < t_2^i < \cdots < t_n^i \leq \tau_2$$

We suppose that $i \in \{1, \ldots, N\}$ and that the N sensors are sampled during the same time period, although not at the same time points. We assume that the times and the measurements are real numbers represented by fixed precision binary representations.

In other words We can represent the collection of measurements $(s^1(\tau_1, \tau_2), \ldots, s^N(\tau_1, \tau_2)$ as a finite length binary sequence. We refer to this binary sequence as the *raw data*.

We assume that the raw data consists of a *signal* which corresponds to properties of the physical system being observed and *noise* which contains no useful information. In practice, the partition between signal and noise might depend on the the goal of the system. However, if computation is unbounded, we can use the concept of the *Kolmogorov Sufficient Statistic* to get a context-independent partition of any binary sequence into signal and noise.

We briefly describe the Kolmogorov Sufficient Statistic, for a more in-depth description see [**?**], page 175. We fix a universal turing machine U. The encoding of the binary sequence x^n is a concatanation of two parts: a program P and a binary string R. The requirement is that the program P, given the string R as input, outputs the sequence x^n, the length of the encoding is the sum of the lengths of the two parts: $l(P) + l(R)$. The Kolmogorov Structure Function $K_k(x^n|n)$ is defined to be the length of the shortest input R such that there exists a program of length at most k which generates x^n upon recieving the input R.

It is pretty obvious that one can always transfer a prefix of length j the input R into the program P, there by making the program longer by j bits and the input shorter by j bits. Therefor increasing k by one bit decreases $K_k(x^n|n)$ by at least one bit, which keeps the total length of the encoding unchanged. Continuing this way until the length of R is zero we arrive at the standard definition of the kolmogorov complexity of a sequence.

We are therefor interested in the shortest program which achieves the kolmogorov complexity (technically: up to a small additive constant). This program is called the *Kolmogorov Sufficient Statistic*. One can say that the program P captures all of the structure of the data sequence x^n while the sequence R captures the unstructured or *random* part.

As we consider data streams, it is natural to extend the two part description given above into a three part description. Our suggestion is that the encoding consists of a *program* P, *parameters* G, and random R. The program P recieves as input both the parameters G and the input R and it generates data x^n. The difference from the prvious definition is that the length of P does not depend on n, in other words, the program captures the part of the structure that is independent of the length of the data x^n. As n increases, G is defined in

the same way as the Kolmogorov sufficient statistic but with the constraint that the first part of the program is P. R is the same as before.

This three-part coding allows us to capture stochastic sequences with non-stationary distributions, hierarchical clustering models, variable length markov models etc.