Approximate Views and compression

May 24, 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 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 reason that lossy compression methods are effective is that not all of the bits in the digital representation of the sensor measurements are equally important. A common model of real-world mesurements considers them to be the sum of a *signal* and *noise*. Where the signal represents the physical quantity measured by the sensor while noise corresponds to the undesirable external and internal influences that cause errors in the measurements. Noise

is often modeled as "Gaussian White Noise" (which can be formalized by Using Weiner processes). A raw signal f is usually represented 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.

It is interesting to note that the Shannon information content of white noise is typically much larger than that of the signal. As a result, most of the bits used in a lossless compression of the raw signal will be devoted to the noise! Lossy compression achieves two things at once: a better compression ratio than lossless compression and a reconstructed signal that is a cleaner version of the raw signal.

The idea of partitioning data into signal and noise, which is common practice in signal processing. Can be generalized by using the concept of The Kolmogorov Sufficient Statistic. We describe this notion in detail below. For this introduction, we present a high level view. Using the method of Kolmogorov sufficient statistic one can represent any signal using a two part code (A, r) where A is a computer program and r is an *uncompressible* binary sequence. In the Kolmogorov view of randomness, an uncompressible sequence is equivalent to a random sequence. We can therefor equate the program with the signal and r with the noise. While changing r will definitely change the reconstructed sequence, it will change it in a way that is *unconsequential* from a statistical point of view. We can therefor encode the sequence s using A and will not be losing any statistically significant information.

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 What is a model?

A model is a mathematical representation of the world. It represents a simplified description of the state of the physical world.

The physical world has a natural four dimensional coordinate system: X, Y, Z and T. A model is a mapping (a function) from (x, y, z, t) to some quantity q. The quantity can be a scalar (i.e. temperature) or a vector (i.e. the energy distribution over a range of frequencies).

Consider, for example, a model of the temperature in UCSD. Many different types of models are possible: A discrete model that partitions space into rooms might be appropriate for indoor spaces while a model that defines temperature as a continuous function of time and space (such as a linear combination of wavelets) might be better for outdoor areas. The time resolution might also depend on the application.

What is common to all models is that the defines the temperature for a continuous range of locations and times. I guess this is what is meant in the following sections by equating models with an abstract "View".

Measurements are a different type of animal altogether. All of our measurements, taken together, corresponds to just a finite set of points in 4D space, and typically the density of the points is far from uniform: some regions have many measurements while others none at all.

Invariably, we will not be able to point to a unique single model. The fewer the measurements, the larger the uncertainty. The more data we have, the more restricted the set of **acceptable** models. The trick is that not all measurements are equally useful. Some sensors malfunction, some have higher noise level than others and some are placed at an unrepresentative location (for example, a temperature guage that is placed right next to a stove). We therefor seek a model that explains *a significant fraction* of the sensor data rather than all of it.

Coming back to the temperature example. Suppose we have a model of all rooms in UCSD that contains parameters that capture the heat transfer between neighboring rooms and between the rooms and the outside of the building. These parameters would be, by and large, fixed for all time. We can then use this model to encode the temperatures in the rooms. The compression rate would be high **but** there will be some rooms where the model does not fit at all and those will not be compressed. The question then becomes whether the consumer needs to be given the noisy signal or whether s/he just needs to be told that this signal is ignored because it is too far from the predictions of the mode. In a world where sensors are cheap but maintaining them is expensive the second option might be better.

Bottom line: I believe that when we measure the quality of a compression scheme we should require is high compression and low error on a *fraction* of the data, and that the fraction be statistically significant, by which I mean, the probability of having this level of compression by chance is extremely small.

3 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. The *raw data* is a database table which we denote by X. Each row in the table represents a single measurement. The raw has time as one of the fields. We denote the *Schema* of X by S.

3. An *approximate view* or a *model* M of X is a representation of a distribution over instances of the Schema S.

4. The model is represented by a program which, given a binary sequence, generates a table according to the schema S.

5. We say that the model is accurate if there is no program that can distinguish between the original table X and a table X generated by feeding the model with a random binary sequence.[1]

[1]This definition depends on the computational model, the ideal is to use the Kolmogorov Definition, but

6. The ability to generate a table that is statistically indistinguishable from the original table is the basic requirements that all views have to satisfy. The basic protocol is: the view generates a randomized table, and the user computes whatever they need to from that table. However, in many cases it is faster to compute the answers to queries directly from the model M, without generating a random sample. The view can provide a mechanism to answer such queries directly.

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).

4 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

that is not practical. A more practical approach would be to use computational models of pseudo-randomness (Russel). The truly practical methods would be domain-specific and will probably use statistical tests or notions of distortion (a-la lossy compression)

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.