Efficient Calculation of Large Collections of Time-Weighted Sums

Mike Koss

Created: June 13, 2008
Updated: Sept 20, 2008

Summary

Time weighted sums using exponential decay are important for comparing time-dependent valuations. Just as net present value calculations can be used to determine the current value of cash flows, so can we use time weighted sums to determine the present value of any stream of valuable actions. For single variable statistics, exponential decay is simple to calculate (see below). But when a system uses a large population of time-weighted sums, which are each updated at different times, and must be compared or sorted, the standard calculation procedure can demand that every variable in the database be updated and normalized to the same time-base.

For a large database, this can demand a lengthy scan and re-write of all the data in the database. In this invention, a method is explained that allows for:

- Simple calculation for updating individually stored time-weighted sums.
- All sums are normalized to a single time-base, and are thus always comparable, can be indexed and sorted, w/o the need for any database scanning.
- Controlling the precision of numbers needing to be stored so that overflow conditions will not arise, and do not necessitate re-normalization of stored data.

In this disclosure I outline:

- The definition of parameters used in calculating time-weighted sums.
- Formulas for calculating time-weighted sums.
- A system for storing time weighted sums that are time invariant, and therefore can be stored in a database for use in sorting and filtering of database queries.

Definitions

Variable	Definition
t	Time (e.g., units may be in days, begining with 0).
h	The characteristic half-life of a value. This *time constant* determines how valuable a past event is compared to a present event.
v(t)	The intrinsic (non time-weighted) value of an event or action (taken at time, t).
S(t)	The time-weighted, net present sum of all actions up to and including time, t.
k	The depreciation factor over one unit of time: $(1/2)^{1/h}$ For example, when h=1, k=0.5. When h=7, k=0.906. $0 < k < 1$

Formulation

Assuming descrete time (t, can take on values, 0, 1, 2, ... etc.), define the value of the stream of actions up to and including time t=n as:

$$S_n = \sum_{t=0}^{n} (1-k)k^{n-t}v_t$$

or, using a difference expression:

$$S_t = (1-k)v_t + kS_{t-1}$$

We apply the (1-k) scaling to normalize the long-term sum of a constant value, v, to be equal to v.

More generally (even for continuous time, t), S(t) can be updated from any score last calculated at time, t1:

$$S_t = (1-k)v_t + k^{t-t_1}S_{t_1}$$

The difficulty in storing values, S(t) in a database, is that they are not time-invariant. They are only relevant at time t. In order to perform queries of the database all values have to be normalized to the same time, which would involve re-writing every record in the database. So, we introduce a time-invariant version of the time weighted sum, choosing time t=0, and escalating the value of all subsequent events.

$$S'_n = \frac{S_n}{k^n} = \sum_{t=0}^{n} \frac{1-k}{k^t} v_t$$

In the absence of subsequent events, the value of S' stays the same throughout time, so stored values need never be updated. A difficulty with this system is in storing these values without overflowing the numeric precision of the underlying numeric data representation. Over a long time period, K^t becomes vanishingly small, and we have to apply exponentially increasing scaling factors to the weights of new events. We can eliminate this problem by instead storing the logarithm of S'. Since logs are uniformly increasing functions, we can similarly compare (and use in query sort orders and indices) values of log(S') to determine relative net present value.

$$log(S'_t) = log(\frac{S_t}{k^t}) = log(S_t) - tlog(k)$$

$$= log_2(S_t) + \frac{t}{h}$$

Note that while S' grows exceedingly large, k^t grows exceedingly small. S'*k^t = S, which is a reasonable sized number for calculation.

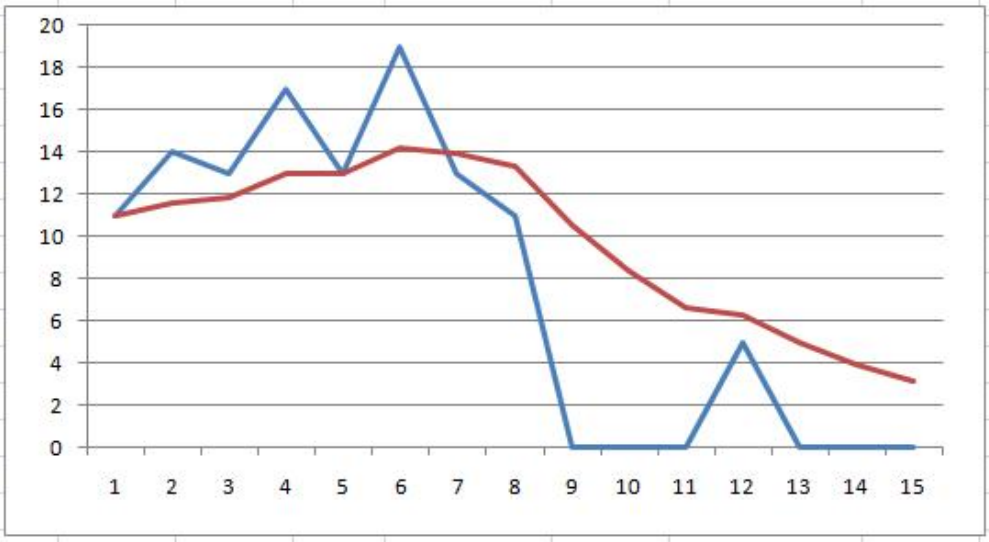
Calculation

To organize the calculation of log(S'), we need store to following values for each activity stream:

1. S - current time weighted sum, at time of latest update
2. tL - time of lates update
3. log(S') - log (base 2) of zero-time based time weighted sum (time invariant)

```
// Untested pseudo-code (JavaScript)

// Time scale constants based on selected half-life time-constant
Score.h = 3;
Score.k = Math.pow(1/2, 1/Score.h);

Score.log2 = Math.log(2);

function Score()
{
   this.S = 0;
   this.LogSp = 1;
   this.tL= 0;
}

// Designed to work even if calls come out of order (t make
decrease between calls)
Score.prototype.Update = function(v, t)
{
   if (t > this.tL)
      {
      this.S = (1-Score.k)*v + Math.pow(Score.k, t-this.tL) *
this.S;
      this.tLast = t;
      }
   else
      {
      // We're getting an old score - don't reset this.tL
      this.S += (1-Score.k)*Math.pow(Score.k, this.tL-t)*v;
      }

   this.LogSp = Math.log(this.S)/Score.log2 + this.tL/Score.h;
}
```

	A	B	C	D	E	F	G	H	I
1	**Inputs**								
2									
3	half life	h	3						
4		k	0.793701	-0.333	log(k) = -1/h				
5		1-k	0.206299						
6									
7		**S(t) = (1-k)*v(t) + k * S(t-1)**							
8									
9	**t**	**v(t)**	**S(t)**		**1/k^t**	**S(t)/k^t**	**log(S(t)/k^t)**		**Calc**
10	0	11	11.00		1.00	11.00	3.46		3.46
11	1	14	11.62		1.26	14.64	3.87		3.87
12	2	13	11.90		1.59	18.90	4.24		4.24
13	3	17	12.96		2.00	25.91	4.70		4.70
14	4	13	12.96		2.52	32.67	5.03		5.03
15	5	19	14.21		3.17	45.11	5.50		5.50
16	6	13	13.96		4.00	55.84	5.80		5.80
17	7	11	13.35		5.04	67.28	6.07		6.07
18	8	0	10.60		6.35	67.28	6.07		6.07
19	9	0	8.41		8.00	67.28	6.07		6.07
20	10	0	6.67		10.08	67.28	6.07		6.07
21	11	5	6.33		12.70	80.38	6.33		6.33
22	12	0	5.02		16.00	80.38	6.33		6.33
23	13	0	3.99		20.16	80.38	6.33		6.33
24	14	0	3.16		25.40	80.38	6.33		6.33



Applications

Time weighted sums are especially useful in applications where events can accumulate value over time, and yet where recent (later time) events have a higher value than older events. For example:

1. Sequences of credits and debits over time into interest-bearing or inflation adjusted accounts.
2. Sequences of events that are indicators of the *popularity* of web sites, movies, books, news stories, blog posts, products, services, etc.
3. Sequences of requests that are made for resource utilization, where a net present resource demand is desired.

An information community is a group of people who share common interests in a given topic and desire to use a social medium to participate in sourcing, ranking, and reading the best available information on that topic.

We develop the techniques of a social scoring system that has the following attributes:

- Ranking popularity of content through group voting
- Categorization of content by tagging
- Automatic adjustment of filter parameters based on user preferences
- Limiting information stream on a "best-available" basis to each user
- Adjustment of quantity of information stream based on each user's consumption rates
- For task-based communities, distribution of work according to ability and availability
- Presumption of time-based relevance and automatic decay of popularity weighting
- Correlation of a user's actions against the norm - Karma ranking relative to topics and users

This system can be used across a wide variety of content types to automatically filter:

- Forums
- Bookmarks
- Micro-blogs/tweets
- Products
- Advertisements
- Email mailing lists
- Search terms and search results

User Gestures

The primary input to a Social Scoring system are the *gestures* that users make in interacting with the information stream they are presented with. The primary gestures this system will consider are:

- Vote up or vote down (request more like this or less like this)
- Silence/block (explicit block of a user, topic, or information source)
- Unfiltered stream request (show all information from a user or source)
- Request to read more (see beyond snippet, click-through of a bookmark)
- Saving/collecting in personal information database (Favoriting)
- Passing along to friends or adding additional topics
- Commenting (to the author or community)

Auditioning

The purpose of a ranking and relevance engine is to filter content so that users are seeing only the most interesting or relevant content. The catch 22 is that if no one sees the content, you don't know how valuable it is. So, its important to have a system of randomly auditioning potential content to the group of possible subscribers. Initial scores are then generated from user interactions with the auditioned content.

Scores should be attributed on a proportional basis. The system should measure the ability of the content to attract valuable gestures based on the number of views (and even based on the placement of those views on a page).

For each information source, there are a community of users who have opted in to that source at different frequency levels:

- Show all (raw stream)
- n/day/week/month limit
- Allow delay (may be old content, but higher ranked)

References

I made a search of Google Patents and Google scholar looking for prior art using search terms:

time decay, scoring, database, exponential

The only related patent I find is the following (ungranted) application:

1. System and method for searching using a temporal dimension**,** http://www.google.com/patents?id=98-WAAAAEBAJ
   I see no mention of storing exponential time decay values in a database - may need further examination.