Twitter as a Sensory Organ:
Analyzing real-time data with a multi-agent system
Peter Sugihara (@_0_)

“The world is it’s own best model.” -- Rodney Brooks

"It would be as useless to perceive how things ‘actually look’ as it would be to watch the random dots on untuned television screens." -- Marvin Minsky

Abstract

Large real-time data sources such as Twitter present rich and interesting sets of situational data. Twitter data affords us a constantly updating and structurally homogenous source for information which often describes the current situation in the outside world. However, with these new living datasets come new challenges. We must analyze data as it comes in rather than performing a single analysis with a fixed dataset. To do so we must design a system which harvests and processes new data as it becomes available. With this in mind, we propose a framework for the development of a distributed multi-agent system which would respond to this challenge by allowing agents to work asynchronously. It’s various parts update a shared collaboration space or blackboard at will. We have built a general model of such a system as well as a rudimentary implementation to which we pose the simple question: “Is it raining in New York City?"
Why multi-agency?

In designing a complex system, the metaphors we start with are of utmost importance [B.1]. There are many real-world examples of systems in which multiple agents collaborate asynchronously towards a common goal. Swarms of bees continuously harvest honey to serve their queen, ants march in droves to, oftentimes cooperatively, carry food back to the mound. Perhaps the best examples of this asynchronous cooperation among humans can be found on the internet where people, often anonymously, cooperate to create encyclopedic stores of knowledge (Wikipedia) and answer challenging questions (Stack Exchange, Quora). We believe that multi-agent systems are so common in nature because they exhibit many positive qualities, two of which I will describe briefly.

First, multi-agent systems are robust against *component failure*. They often respond surprisingly well when a part of the system simply goes away. For instance, when one ant dies, the system marches on and food collection continues as if nothing had ever happened. This is in large part an attribute of subsystem redundancy but it is also indicative of a much deeper quality of *distributed control* which will be dealt with in more detail later. In short, there is no single most important point.

Secondly, and this is tied closely to the first reason, multi-agent systems are generally good at handling change. It is assumed that the environment will be dynamic. Beyond component failure, ideal multi-agent systems are able to restructure themselves in the face of environmental changes. For an example of this in nature, we may consider the pheromone trails of ants. When an ant is successful in finding a piece of food, it leaves a temporary trail of pheromones on it’s way back to the mound. Other ants may then follow this trail. When a path to food is blocked or a food source has been exhausted, the ants will split up to look for a different food source, leaving a trail back to the mound if one is found.
These types of systems are able to reorganize themselves without any central planning mechanism.

For these reasons, a multi-agent system seems ideal for the task of analyzing a constantly growing and ever-changing dataset and there is a large and growing body of work to indicate as much [A2, A3]. For inter-agent communication we look to the blackboard metaphor for inspiration, wherein a group of experts with different specializations collaborate by posting and responding to hypotheses in a shared memory space [A.1, A.1a, A.1b, A.1c]. Often these systems have contained some notion of a controller, a boss entity (sometimes built into the blackboard) who dictates the order in which agents may examine and post to the board. However, some of the most common human manifestations of this type of virtual blackboard, such as web forums and wikis, are very loosely moderated and agents (readers and authors) act asynchronously, without direct control. The web server seems almost purpose built to be a type of digital asynchronous blackboard.

There are notable advantages to this type of asynchrony, especially in an environment where one can assume that agents will act cooperatively (unlike message boards which must protect against spam attacks from malicious users). For one, we do not require that a “controller” agent knows about all other agents. In fact it eliminates the need for any direct agent-to-agent communication. Inspired by the way many web applications fake updating in real time, agents periodically check the shared evidence space for changes. If a change has occurred, they may respond by posting to the evidence space at their leisure. In this way, each agent only needs to understand the pieces of evidence which they are concerned with. For instance, an agent—we’ll call him Jenkins—who does time-series analysis must identify which piece of data is a time-series but does not have to understand how the time-series was generated or what it represents. Hamilton would simply
check the board for a time-series and if it contained one, output an analysis tagged to indicate which time-series it corresponded with.

By designing our system to act asynchronously, we also simplify the method by which it can be distributed across many processes or machines. The evidence space simply becomes a shared database which each agent may query at their leisure. Furthermore, when we wish to add additional agents, such as an expert who we’ll call Box to make a conclusion based on Hamilton’s time-series analysis, we don’t need to modify Hamilton at all. We need only to tell our new agent, Box, where to find the evidence space and the format of Hamilton’s output. Hamilton and Box will never need to directly communicate or even know each other’s name despite the fact that they are in fact collaborating.

It should be noted that asynchronous systems do have a large drawback in that they are, by nature, less predictable and it can be harder to reason about the system as a whole. However, it is our claim that their benefits are large enough to make them a viable option.

**Implementation**

We have completed a rudimentary implementation of this type of system written in Python using a Redis key-value store for the shared evidence space. Our system contains 5 classes of simple agents and tries to answer the question, “Is it raining in New York City?”. Our five agents are as follows:

**Tracker**

The Tracker pulls in data from the world to the evidence space. It handles a tweet stream for a list of keywords using Twitter’s Streaming API. When connecting to Twitter, it adds a location word to each keyword, in our case “NYC” and “Manhattan” are both added for each
keyword. These tweets are posted to the blackboard along with a default confidence score for each.

Arbiter

The Arbiter does maintenance on the evidence space. It periodically penalizes all confidence scores, the assumption being that older evidence is less important to the problem. It also removes evidence with confidence scores of 0 so that our evidence space never grows to an unmanageable size.

Expert

The Experts have domain knowledge about a hypothesis and check if the evidence indicates a hypothesis. They are initialized with a set of keywords which they look for in the evidence. They perform checks of the evidence space periodically and print their results. Our system has two experts, one which identifies evidence that the weather is rainy and another which identifies evidence that the weather is sunny.

ReputationAdjuster

A ReputationAdjuster boost and penalizes the confidence scores for evidence from reliable sources. In this case, we track reliable Twitter user ID's and boost their scores at intervals.

KeywordSuggester

A KeywordSuggester suggests new keywords based on word collocation within tweets. This type of association learning could be incorporated into the ReputationAdjuster to find new reputable tweeters as well. The suggested keywords are placed in a set in the datastore
which is checked periodically by the Experts and Tracker. A description of the learning mechanism is given below.

These agents run in parallel, each on a different thread, and connect to the shared Redis evidence space running on localhost. Because the agents only interface with the evidence space and not each other, this architecture could easily be distributed across multiple machines by placing the Redis datastore on a server with a static IP address. Each agent is instantiated with an interval at which it runs its respective task.

A typical sequence of moves in our system is shown below:

1) The tracker alerts Twitter’s Streaming API that it is interested in a set of keywords.
2) Some time later, Twitter responds with a Tweet.
3) The tracker identifies it as containing the keyword “storm” and places it in a set of “storm” Tweets in the evidence space with an attached confidence score of 10. Though not pictured, it is also placed in a set of Tweets by this particular author which the ReputationAdjuster may query.

4) Every 7 seconds, the “rain” Expert checks each evidence set with a “rain” keyword and sums the scores.

5) It then posts the score to a log list in the evidence space. We could easily add an additional agent which utilized this score to do an additional calculation.

Little bags of words

We analyze tweets using a “bag of words” approach wherein we simply look at word frequency rather than trying to parse the sentence. The assumption here is that due to the 140 character limit, a single Tweet will be about a single subject. For instance, if someone mentions “rain” and “NYC” in the same Tweet, we assume that they are talking about rain in New York City. Additionally we take into account the general trend that tweets are literal status updates in that they pertain to the present moment and, in general, the present locale of the author. These assumptions are sometimes violated though they seem to be true in aggregate.

One such violation can be seen in the following bit of output. The system is tracking 23 keywords and initially the “rain” and “shine” experts are both outputting scores of 0. The tracker receives 3 tweets in succession and places them into their respective keyword sets in the evidence space. The first 2 are quite reliable and are written by a weather update service. The third highlighted tweet is received because it uses the word “storm” yet it does not pertain to weather at all. Each are placed into the evidence space and given the same
confidence rating. The rain expert sees the all three tweets and because the system does no
sentiment analysis, factors them equally into the score of 30:

rain  shine tracking
0   0   23
Found evidence with keyword “rain”:
Forecast for NYC Tuesday: Rain. High temp: 52F. #OWS #tpp #tpp
Forecast for NYC Tuesday night: Rain. Low temp: 50F.
#OccupyWallStreet #tcot #tlot
Found evidence with keyword “rain”:
Best news ever?! @SummerRose and I are going to be taking NYC by
storm. #NYU
30.0  0   23

There are two methods with which future implementations could deal with this type of
imperfection. The first would be to add some sort of topical analysis feature to the expert.
However, if we wanted to continue to track this flawed score while we added more
functionality (one might imagine some other more critical application where we would need
to keep a mediocre agent running) we could simply add another type of expert to the system
which would intelligently penalize confidence scores similar to the way our
ReputationAdjuster penalizes boosts scores from reliable authors.

Imperfect learning
In an effort to produce a larger dataset and more frequent updates in our system, we
developed a method by which experts may learn additional keywords. When the evidence
space contains evidence for each hypothesis, we look for words which seem to characterize
one hypothesis but not the other. This is carried out by first tokenizing the words in each
hypothesis’ evidence set (named “hypothesis:<hypothesis name:evidence”) and removing
words which occur in both sets. We then recommend words that appear multiple times for a
particular hypothesis by placing them in a set named “hypothesis:<hypothesis name>:keywords”. This logic is encapsulated within a KeywordRecommender agent.

Unfortunately this agent is highly susceptible to error, particularly if the word frequency threshold for adding a word to our vocabulary is set low. This can be seen below where a Tweet about the weather at a protest leads to “occupywallstreet” being added to the tracked keywords. This error cascades until we finally have a large number of keywords which have nothing to do with our weather question (highlighted in yellow):

rain shine tracking
10.0 18.0 23
8.0 15.0 23
8.0 15.0 23
6.0 12.0 23
4.0 9.0 23

Found evidence with keyword: clear
Forecast for NYC Friday night: Mostly clear. Low temp: 45F. #OccupyWallStreet #p2 #ocra

Found evidence with keyword: sunny
Forecast for NYC Saturday: Sunny. High temp: 58F. #OccupyWallSt #p2 #teaparty

Added to hypothesis:shine:keywords :
[['occupywallstreet']]


When do we reach the goal?

If only life were so simple. Goal states are extremely context dependent. In our example, we might wish to know how the weather is right now, how it will be in 5 minutes, 1 hour, or 1 day. Each of these questions requires slightly different analysis and slightly
different criteria to be met. For this reason, we have kept our system free of an inherent goal state. Instead, one may check the state of the system at any time and make inferences based on active agents.

One can imagine creating a simple agent to automatically tell us when we’re done. For instance, if we wanted to know if it was raining at the moment, we would create an instantaneous weather checker. It could simply compare the latest outputs of the “rain” and “shine” experts and give some probability estimation of whether it was raining at that moment. However, if we wanted to check whether it would rain the tomorrow, we would want to create a more complex agent which would do some type of forecasting based on a series of outputs from the sub-experts.

Conclusions and Future Work

That our system generally indicates a correct answer to our question is of little note. In essence we are simply counting word frequency on Twitter and there is little sophistication in the method we use to expand the system’s vocabulary. Rather, we have purposefully rejected the use of complex natural language processing techniques at this point to highlight what we feel is novel about our architecture. Namely, we have created a society of agents which have isolated knowledge but collaborate and communicate asynchronously. Moreover it is our claim that this architecture is easily extendable and fully scalable to hundreds of agents. The Redis in-memory key-value store running locally on a 2010 MacBook Pro with six agents reading and writing to it for hours used less than %0.1 of the CPU time and less than 1MB of memory! Of course if any serious time-series analysis were to be performed or if the system were to run for an extended period of time, a full-fledged database system such as MySQL or MongoDB would need to be added to preserve extended system logs. Of equal importance to the technical scalability, is the scalability of the mental load incurred by
each additional agent. Again, it is our claim that our system of one-way communication allows the programmer to reason about a single agent at a time, evaluating that agent’s impact on the environment (the evidence space) in isolation before adding it.
Excerpt from a successful run on a rainy day:

rain  shine tracking
12.0  1.0  23
Found evidence with keyword "overcast":
"Forecast for NYC tonight: Overcast. Low temp: 42F.  #OWS #teaparty #GlennBeck"
Found evidence with keyword "rain":
"Forecast for NYC Tuesday: Rain. High temp: 52F.  #OWS #p2 #GlennBeck"
32.0  1.0  23
27.0  0.0  23
27.0  0.0  23
22.0  0  23
22.0  0  23
17.0  0  23
14.0  0  23
14.0  0  23
11.0  0  23
11.0  0  23
8.0  0  23

Excerpt from a successful run on a warm night:

Found evidence with keyword "clear":
"Forecast for NYC Thursday night: Mostly clear. Low temp: 42F.  #OWS #tpp #TopProg"
Found evidence with keyword "sunny":
"Forecast for NYC Friday: Sunny. High temp: 58F.  #OWS #TopProg #tcot"
8.0  24.0  23
6.0  21.0  23
4.0  18.0  23
4.0  18.0  23
2.0  15.0  23
2.0  15.0  23
0.0  12.0  23
0  10.0  23
0  10.0  23
0  8.0  23
0  8.0  23
Found evidence with keyword "overcast":
"New York, New York Weather :: 58F OVERCAST: 58F OVERCAST #Weather #NYC"
10.0   6.0    23
Found evidence with keyword "clear":
"Forecast for NYC Friday night: Mostly clear. Low temp: 45F. #OccupyWallStreet #teaparty #ocra"
Found evidence with keyword "sunny":
"Forecast for NYC Saturday: Sunny. High temp: 58F. #OccupyWallSt #sgp #sgp"
10.0   26.0   23
Found evidence with keyword "warm":
"@___KESH sending you positive vibrations on this warm NYC night ((&lt;3))"
9.0   32.0   23
8.0   27.0   23
8.0   27.0   23
7.0   22.0   23
7.0   22.0   23
6.0   19.0   23
5.0   16.0   23
5.0   16.0   23
4.0   13.0   23