SemanticSky

Taking the network up to heaven.

Contents

Part I
Starfish.

1 (The previous project)

2 (Results)

Part II
SemanticSky.

3 Overview of Semanticsky.

SemanticSky (currently) is a program which, given as input a set of documents, outputs a small multi-agent system where some built-in algorithmic agents try to infer similarity relations between the input documents.

The main steps through which this is achieved follow:

- First, each document is transformed into a **Cloud**: an object which theoretically should encapsulate all the information we can extract from the base document. Examples include: names of people mentioned in the document, names of places, statistically frequent word and frequently co-occurring pairs of words.
- Secondly, these clouds are pairwise evaluated by a list of algorithms, called **Guardian Angels** which try to judge their similarity based on different and possibly complementary criteria.
- Finally, the evaluations of the Guardian Angels are merged into a single one, which represents the final state of the system, at this stage.

When the system is first initialized, or when a new document is added to the corpus, this is all there is to it.

But then, some more things will happen, as **Agents** (true people) evaluate by themselves the appropriateness of a link or the relevance of a tag or keyword to some page.

- The current state of the system is kept under watch on by a supervisor algorithm called **God**, is judged either stable or unsettled. If some part

of the network is unsettled, then God waits for more discussion to happen between agents and maybe even fuels it by suggesting relevant links to the parties involved or informing the parties of the presence of a third option, and so on...

- Once the system settles on some decision, such as the complete relatedness of two items, or maybe even the plain equivalence of two approaches that just happened to go under different names, then God gives feedback to all agents involved (human or algorithmic) regarding the accuracy of their guesses relative to the final state. This way, suggestion-givers who gave bad suggestions will be taken less into account in the future.

Crucially, the supervisor algorithm will need to 'suspend judgement' on user-backed agents while a discussion is still going on, so that plain democracy will decide on the final state of the system (the objective is to respect people's suggestions, not to distort them) without being influenced by the feedback they received. God is not a moderator: is a plain container and merger of opinions. He listens to the discussion and takes notes. On the other hand it's up to him to judge whether an algorithm is good or not.

Currently it's very easy to implement a new algorithm in the system:

```
import clues
def evaluate(cloud1,cloud2):
        ...
        return myevaluation

myalgorithm = clues.algorithms.Algorithm(evaluate)
myguardian = clues.GuardianAngel(myalgorithm)
myguardian.evaluate_all()
# this will make it evaluate
# all 2-permutations of clouds.
```

But not all algorithm are good, and some are good just on very specific subsets of the system. Suppose that we have an algorithm that just checks whether the authors of the document are the same person (and does that by regex matching some query). This algorithm will then return zero for all documents where an 'author' is not defined or the query doesn't match. God then will (try to) understand that the algorithm is useful in some subdomain, even though his average accuracy is very low.

The evaluations that a Guardian Angel performs differ from those of an human agent only in that they are produced automatically. Their route is

in fact basically the same: they either get queued and later on processed by God, or (by default) they get immediately processed.

The 'processing' implemented is currently quite naive. When an agent has a suspect strong x about the similarity of two documents $y = sim(a, b)$, a "Clue" is spawned that conveys the information that the agent has an opinion x about the fact y.

Once the clue gets read by God, he will log it and then retrieve all logs about y; that is, all the information he has about y. Then, he will retrieve all the confidences of the clues (how strongly who produced them believed into them) and the weights (how well their authors usually do at guessing).

The weighted confidences are then averaged, and God's belief about x is updated to the thus obtained value.

Once the system reaches a stable situation about some fact y, the supervisor algorithm will hand out feedback to all the algorithms which had some suggestions about y or the discussion surrounding it. We can in fact imagine that, taken any pair of documents, there will be some arguments in favour and some against their similarity or relevance to one another. Not unlike what goes on behind a Wikipedia page (in the 'discussion' section), people will be allowed to debate about various issues, such as pertinence of tags, of links, and in general, let us say, the structure of the network. Once the discussion is over (this can be detected by, for example, a month of silence in the discussion page), the system will then adjust the weights of the Guardian Angels' future suggestions. For example, suppose that an algorithm gave particularly good suggestions in this situation (but not in many others); God will then try to guess what is that the algorithm (and maybe even the agent) is good and bad about, and adjust his weights selectively, not unlikely what goes on in the so-called Stacked Generalization models and Mixture of Experts models, which will be discussed later in the Supervisor.Feedback section.

The following sections will try to show in some more details how SemanticSky currently (as of the end of July, 2014) works; starting from the clouds, up to the very heart of this tiny digital pantheon.

4 Clouds: the bricks.

At the moment, clouds are more or less simple wrappers for documents. The central data structure for a cloud is what we will call **layers**. What makes it a simple wrapper is the fact that only the first layer is currently being filled; but the main idea behind a layered cloud is that it should include

hierarchically ordered information: from the most precious (and most hardly one-to-one matchable) information to the second-hand, less polished one.

To show the reason behind this few things would do better than an example: suppose we have a document called 'Rise and Fall of Ziggy Stardust', containing a couple of pages of history of this man called Ziggy. Unless the title is ironic or in any way misleading, which, assume, is not the case, we already have in front of us a few very important informations: that this document is about a man called Ziggy Stardust and that a rise and a fall are somehow involved.

SemanticSky currently performs little or no semantic analysis, so the cloud will not know that it is Ziggy rising and falling, but just that the names 'Ziggy', 'rise', 'fall' are relevant descriptors of the document. And what's so important about the title is that the mere fact of it being a title tells us that the information stored there is important. This is possible, of course, only because the input data is partly annotated: it's not an uniform body of plain text, but already contains some extra information which we can use. In absence of this, other methods could be used to extract headers, titles and other kind of sources of usually very relevant information.

This is what layers are meant to be for: the innermost layer will certainly contain these four words, plus the most relevant others that, with diverse heuristics, we will be able to extract from the body of the document itself.

Currently, the clouds store just about everything in a single layer, as the information we have is all first-hand: comes directly from the document, which is the most trustworthy source of information we can have at the moment. The layer-building function currently tokenizes everything down to words, stems them (so as to capture the relevance of, say, 'learn-', even though in the document the concept appears under many different grammatical forms, which would make the statistical relevance of 'learn' by itself very low) and finally produce a list of the pairs of words most frequently co-occurring in sentences.

The sorts of information a cloud currently contains in its only layer are (extracted via regex matching and tokenization):

- names: all Capitalized Sequences Of Words.
- urls: all urls either hidden in hrefs (the documents' text is html) or explicitly mentioned in the text.
- words: information about their frequency (tf) *and* their frequency weighted by their inverse document frequency (tf-idf).

- core: a special subset of words which we have reasons to believe more important than the other ones; namely those which come from titles or which are labeled as 'headline'.

- tags: a list of the tags assigned to the Starfish item.

4.1 Future work.

The framework can clearly be expanded even at this level, and time permitting, this will be most certainly one of the most promising directions to go. The layer-based structure is already there, but is not currently used. An immediate expansion of SemanticSky could involve filling the lower-level layers.

Possible sources for the information to fill these layers with include:

1. The web. The second layer could be filled up, for example, with information drawn from google querying for 'The Rise and Fall of Ziggy Stardust' or some other keywords.

2. Other clouds. At some stage, suppose, the system will settle on the decision that Ziggy's cloud is clearly related (for most of the Starfish users, at least) with (say) David Bowie's cloud. Then, once the confidence about this fact is higher than a certain threshold, we might let some keywords of either clouds 'filter' into the lower layers of the other cloud.[1]

3. Corpora information. From a corpus search we might discover that 'stardust' is very frequently related with Carl Sagan and stars.

The latter example about stars and Carl Sagan is a clear situation where we want to make sure that this information is taken as second-hand only and is given appropriately less weight than the first-hand one.

5 Guardians.

Guardian Angels are currently the core of the algorithmic part of SemanticSky, and the interaction of them, the Agents and God is probably the theoretically most interesting thing going on there.

[1] This might have the side effect of forming loops of self-reinforcing feedbacks, so we will have to make sure that algorithms, when re-evaluating the two clouds, won't take into account *that* information as well.

Basically, a Guardian is nothing but an agent that only examines pairs of items when prompted (by God) and that takes decisions based on a never changing algorithm.

Their strength is of a collective kind: each of them takes decisions based on a rather small part of all the evidence available, and produces a generally inaccurate (but, on average, above chance) prediction.

Follows a quick description of all the algorithms currently implemented (which is basically their docstring).

tf and tf_idf_weighting The most standard Guardian Angels, just retrieve the tf / tf-idf value for each word in the bags of words of the documents (also held in the clouds) and returns the cosine of the angle of the two vectorized documents.

naive_name_comparison Based on the 'names' section of the layers of the clouds, which were previously extracted via regex matching, this algorithm just checks how many names the two clouds share, and normalizes it against the set union of the two lists of names. The required kind of matching is very strict: a 'Ziggy Stardust' in a cloud won't match a 'Ziggy, S.' in the other. This could be improved in many ways.

Plus, a name is currently Whatever Is Capitalized, and this is not really so. We are lucky that Starfish does not have pages in German.

Given the little information the algorithm can work on, the 600 nonzero results against 210000 is a reasonable output. Interestingly enough, the algorithm is one of the most precise ones, with a 15% chance that if he detects something, there is actually something.

extended_name_comparison This algorithm, as all other algorithms containing the word 'extended', is based on the social network principle that if A's friends are B's friends, then A and B are likely to be friends themselves.

Let $names(\phi)$ be the set of names contained in the zero layer of the cloud ϕ. Then, for $\phi \in A, B$, we define

$$names(\phi) = \phi.layers[0]['names']$$

$$Neighbours(\phi) = \{a | a \in sky\ if\ names(\phi) \cap names(a) \neq \varnothing\}$$

Finally, we make an extended bag of words by summing ϕ's names with the cores of all of ϕ's neighbours: that is, the most relevant words for all of the clouds which share at least a name with ϕ. Then, we count the overlaps and normalize against the union of the sets.

Not as precise as naive_name_comparison and much more computationally heavy, but gives nonzero results more often.

naive_core_overlap The same as naive_name_comparison, but instead of the 'names' entry of the clouds' layers, takes the core, performing a simple $len(intersection)/len(union)$ computation.

extended_core_overlap let $C(\psi)$ denote the core of (the zero layer of) some cloud ψ. This time, unlike the extended_name_overlap algorithm, the words to extend $C(\psi)$ are going to be drawn not directly from other clouds, but from a global repository of the co-occurrence counts of words in sentences based on the whole corpus.

So, the globally most frequently co-occurring pairs, not weighted by idf (which could of course be an interesting extension), go to extend $C(\psi)$ if at least one of the words in the co-occurrence pair is in the core of ψ.

So, the extended core of psi $E(C(\psi))$ will be, where $coo(a, b) \iff a$ and b co-occur more than twice in the whole corpus, and W is the set of all words occurring in the corpus:

$$E(C(\psi)) = w | w \in W \wedge (w \in C(\psi) \vee \exists b \in C(\psi) : coo(w, b))$$

Finally, the two extended cores are compared in the usual intersection / union way.

tag_similarity_naive This algorithm, as well as the following ones, are made specifically for evaluating the similarity of tag clouds: clouds that wrap tag-type items in Starfish. Typically, the only information we have about this kind of items, beside the name of the tag itself, is a list of synonyms (aliased tags) and perhaps a glossary (a small document). This makes their comparison rather tricky for other algorithms.

Furthermore, there currently are no links in Starfish between tags and other types of items, so we don't have a training set to assess the performance of these algorithms. The choice of treating tag-type items like all other types may also be questioned, but this is not the place for it.

The algorithm, given two clouds, just counts how many pages are tagged with both tags (that is, the tags that the clouds are built around) and normalizes it against the length of the union.

tag_similarity_extended This algorithm is an instance of a higher-level Guardian Angel, or Meta Angel: to work, it needs some previous evaluation.

Basically, it measures the overlap of the sets of clouds marked with the two tags and returns an averaged confidence of the relations of these links.

Suppose we have the following situation, where a, b are tags (i.e. strings such as 'LearningAnalytics','DavidBowie'):

$a \rightarrow$ [cloud1, cloud2]; that is: both cloud1 and cloud2 are tagged with 'a'.

$b \rightarrow$ [cloud1, cloud3]

$c \rightarrow$ [cloud4, cloud5]

Clearly, similarity(cloud1,cloud1) = 1, so tag a and b will be at least 0.5 related since they share half of their clouds; more if cloud2 and cloud3 are related in gods' beliefs. Then suppose cloud1 and cloud4 are believed to be related by 0.4, and cloud5, cloud2, are related by 0.6, but cloud5 and cloud3 are totally unrelated. Then, tag c will be much closer to tag a than to tag b.

The algorithm just computes over this intuition.

tag_overlap This algorithm, unlike the two previous ones, is a measure of the distance of two nontag-type clouds, and simply measures the intersection/union of the tags of the two clouds.

coo_dicts_overlap This algorithm has two versions, plus a third one which consists simply in taking the best result from v1 and v2.

v1 This version returns a pair-to-pair correspondence check of co-occurring pairs of words in the two clouds. Such correspondence is very valuable, but rare.

v2 This version is more permissive (returns many more nonzero values) and consists in splitting the pairs thereby using single words as term of comparison. (Basically it performs a bag of words overlap/union, building the bags from the co-occurrence dictionary). Implements Grefenstette (1994) algorithm for computing the values.

coo_dicts_neighbour Taken the co-occurring pairs in cloud ϕ, denoted $coo(\phi)$, we split them and augment this bag of words with the most frequent words that often co-occur with them at the whole corpus level. (Similar to extended_core_overlap), but based on a different starting set. Then compares the two extended bags.

coo_dicts_extended_neighbour This algorithm is very complex, takes ages to run and is so permissive that it captures far more noise than signal. Probably garbage.

5.1 Future work.

- Guardian Angels are cool, but they also are currently very stupid. Each one of them just performs a little task in a probably very naive way: this could be improved (but, on the other hand, the main idea behind ensembles is precisely a large number of very stupid algorithms that collectively produce a very smart decision. Be it their small number or their being too stupid, currently this doesn't happen so much).
- Plus, the more and most diverse the angels' decision algorithm are, the higher the overall performance of the system will be. Thus, one immediate way to extend the system will be to implement more guardians, or to add slight variations to the already existing ones.
- As noticed above, also, the decisions of the algorithms which are currently implemented are based on a rather small subset of all the available information. Some algorithm only takes into account the words' frequency, some other the words' idf-frequency, some other just the 'names' appearing in the text, and so on. Some effort might be put into smarter algorithms able to combine on the fly all these different types of inputs.
- Another thing which might change is that currently algorithms don't (strictly speaking) learn anything. They just go on spitting the same decisions over and over, and is a higher-level algorithm that is delegated the work of gating their decisions so that they get their weight appropriately with respect to their usual worth. An attempt could be made to include amongst this kind of algorithm some more standard learners, such as neural networks or web crawlers[2].
- Finally, the Guardians' reach could be much extended by having them evaluate not only pairs of clouds but also single clouds, or larger groups. An algorithm for example might try to find hubs in the network, or islands that might then be addressed by 'bridging' attempts. Suppose for example that there is a rather insulated part of the network that is rather inaccessible from

[2]A first attempt in this direction has already been made: I constructed an algorithm which would, given a pair of words a, b, retrieve the number of google hits for the query "NEAR(a, b)—NEAR(a, b)", later normalizing it with the number of hits for queries containing just the single words a and b. To do the same with clouds is not as straightforward and will need some more research and might involve, for example, keyword extraction.

the external world. not only that is an interesting information by itself, but also we may want to fuel discussion by, for example, suggesting more links to 'external' clouds or by giving incentives to those who propose such links.[3]

6 Agents.

7 The Supervisor.

8 (Results)

Currently there are two tests of the framework on line.

The first one consists in removing all the clouds from the sky and initializing an empty god on such empty sky. Then, we add back one by one[4]

Then, after adding each cloud, we ask all the guardians to evaluate the whole sky; that is: to assess all similarities they can spot between the present clouds. Finally, the Knower (an algorithm that 'knows the truth') judges on their evaluations and triggers feedback.

The second test consists in starting with a full sky, but with no feedback. Then, one by one, we give feedback to each item in gods' beliefs, thus (through feedback) adjusting gods' weights and the trustworthiness of angels, refining the beliefs at each step.

8.1 Test 1: online evaluation.

First I will write down clearly and discuss, where needed, the parameters I used for the test; then show the results.

- Learning rate = 0.2.
- Punish = False. This means that guardians don't receive negative feedback for links that are not in the knower's database of 'truths'.

In a real-life application of Semantic Sky this will be a nonexisting issue, but in a test, we have to choose which items we give negative feedback to.

[3] A sketch of higher-level Guardian Angels, which I called Meta Angels, is already present but has currently not been tested or used. In this case, the meta angel tries to use evaluations as they come out of the lower-level angels and take them further: given two items x, y the meta angel's evaluation is a function of how many neighbours x and y share (weighted clearly by how near they are) in the current state of the system.

[4] Or, to reduce the running time of the test, a bunch of them per time. In fact, the tests I ran mostly were made with step 5 and 6. This has also the effect of reducing the amount of memory used (by $1/n$th) for n = step. And given that for step 6 it's still $74mb$, this is necessary.

In this test I chose not to give negative feedback for all links currently not in Starfish and mistakenly captured by the guardians as similar. Maybe in the future it will be interesting to see how the outcome changes setting this flag to true.

To state this more clearly: guardians receive feedback only on items in the knower's database = Starfish's database. The valuation they get for some link, namely, is 1− their current rating of the link.

- Step = 6: clouds were added back to the sky 6 by 6.
- Update rule = median of all feedbacks. The current trustworthiness of some angel relative to some link type (person-to-person, question-to-answer...) is the *median* of all the feedback he has ever received about links with that link type. Other available solutions include averaging (sum(all feedbacks)/ len(all feedbacks)) and step-by-step averaging (value = (previous value + value) / 2).
- Merging rule = standard merge. This affects how the new suggested value (as computed by the update rule) is merged with the previous one. Extremes include ignoring it and replacing the latter with the former. The standard merge is:

$$new\_value = previous\_value - [(previous\_value - new\_value) * learning\_rate]$$

This is, at each loop of the test, the average value of each agent's beliefs in all true links; that is, all the links that according to our training set (represented by the knower) are there.

We can see how the values increase and (seem to) stabilize after around 45 iterations. This means that there were 270 clouds (since the step was 6) in the sky at that point.

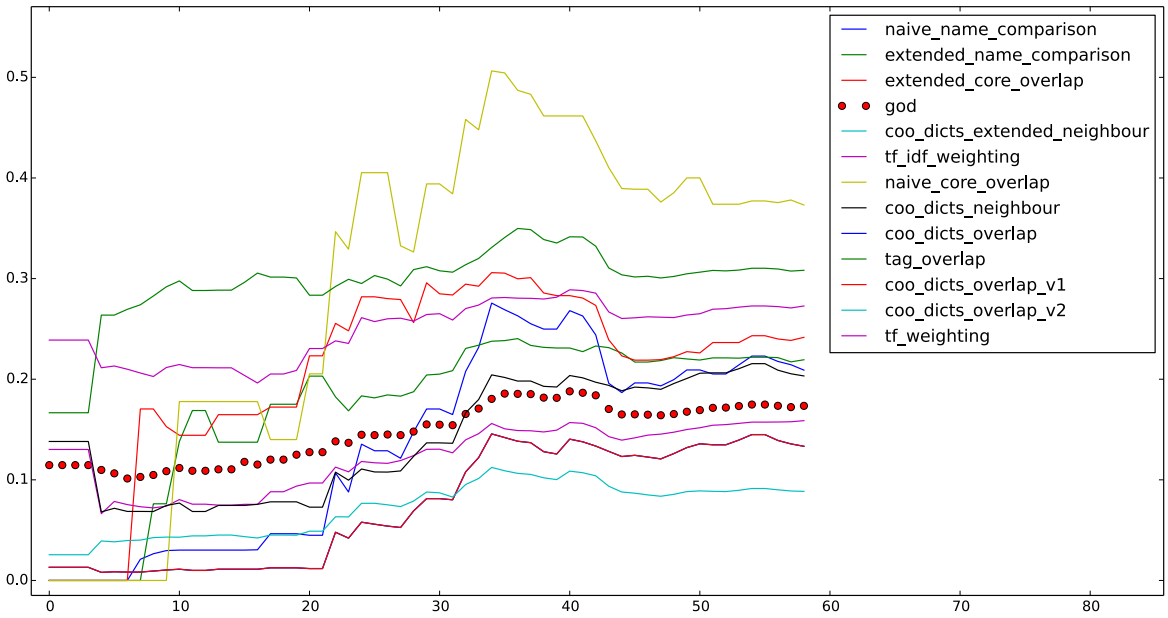

Figure 1

Figure 2 shows the progression of average beliefs in false links: that is, links that are currently not in Starfish. As the test proceeds, the values quickly stabilize around (an average of) 0.05.

The convergence is much faster because there are many more false links than true ones.

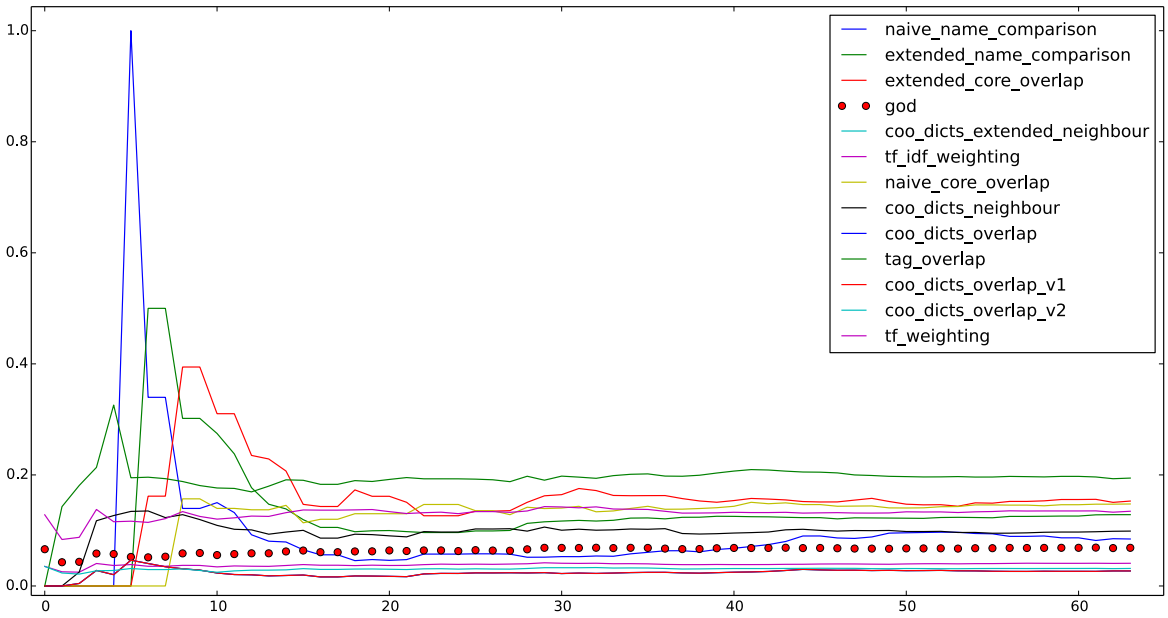

Figure 2

Finally, these are the (non-cumulative) regrets of all angels and God itself.

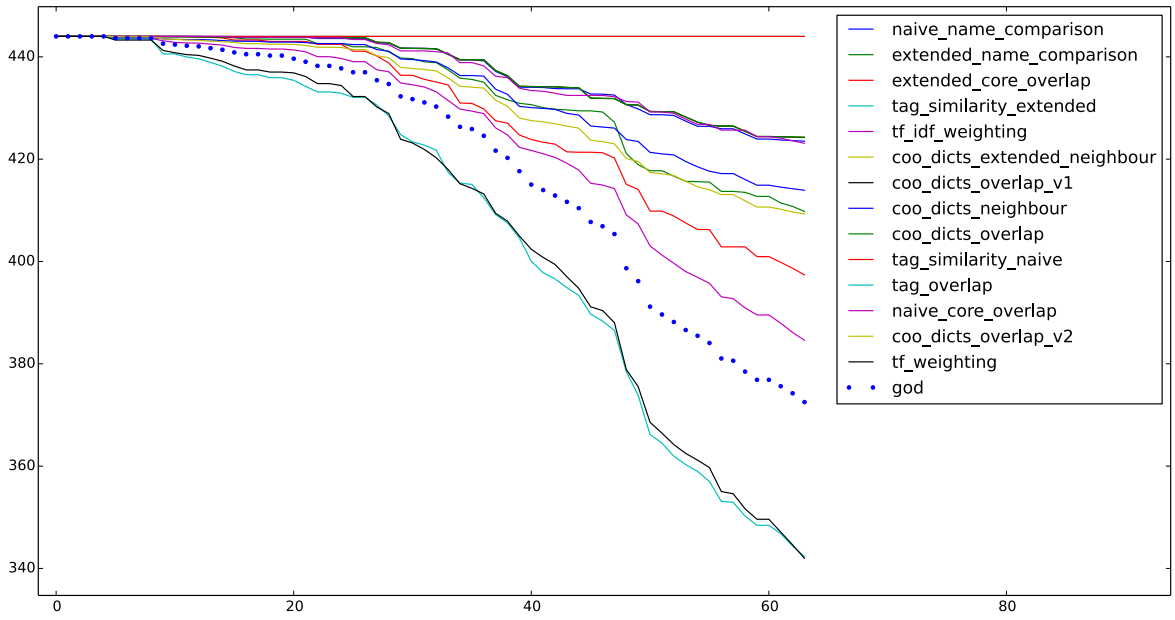

Figure 3

Regret (at stage n, for angel A) was computed as follows: $sum((1 - A.belief\_in(x))$ for x in all_truths), where all_truths is a list of all links actually existing in Starfish (that is: the Knower's database).

The angels whose regrets don't decrease are the tag-similarity angels (hidden for clarity's sake in the next plots). Currently lacking Starfish any data about how similar two tags are, no link of that type (tag-to-tag) is available in Starfish, and all the feedback will be negative.

This also means that their trustworthiness (relative to the only type of link they can clue upon) will always tend to zero very quickly.

One nice thing to notice is that all the algorithms' regret increment is stably decreasing: there is no faulty learner.

One can also notice that god's line is not the simple average of the other lines, but is a weighted average: it will be influenced (negatively) proportionally to the regrets of the angels: an high-regrets angel will probably have low average weights, and thus influence less god's beliefs (and thus, god's regrets).

A few extra facts :

The same test, with different parameters (learning rate = 0.8, update

rule = *average* of all feedback) yields comparable results (though perhaps a bit less nice):

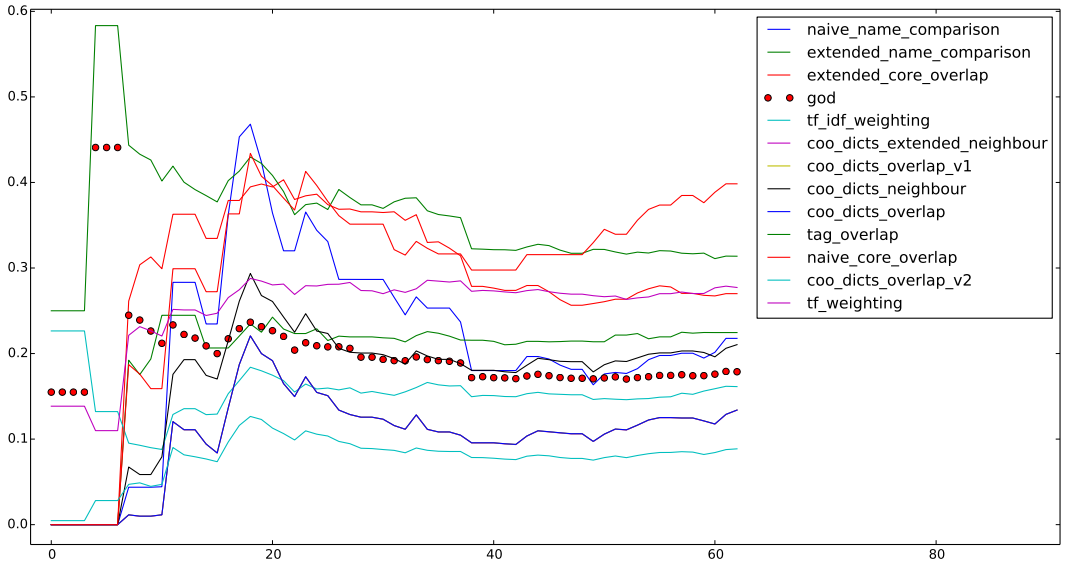

Figure 4: true links beliefs

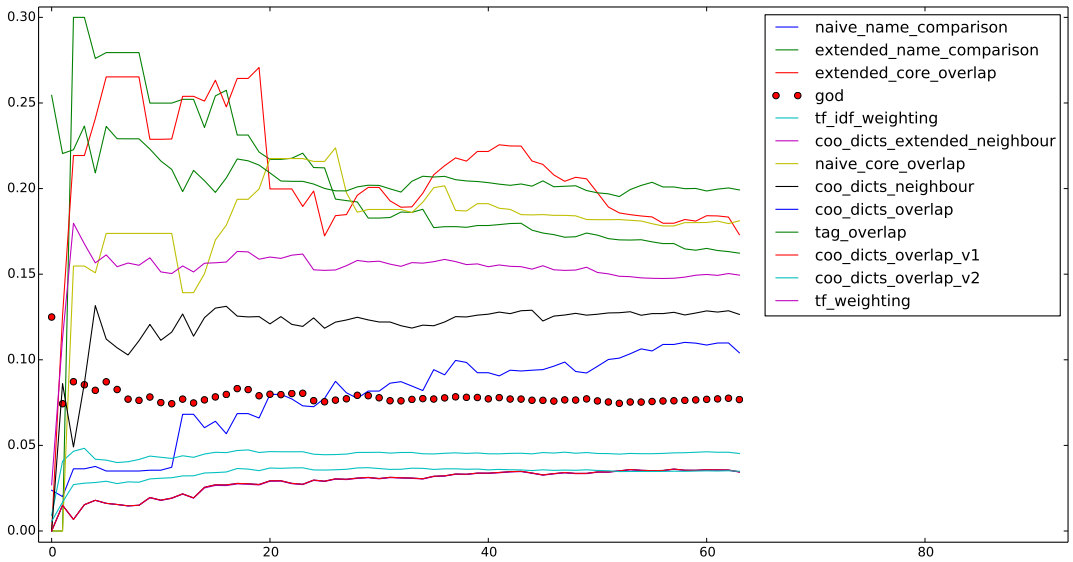

Figure 5: false links beliefs

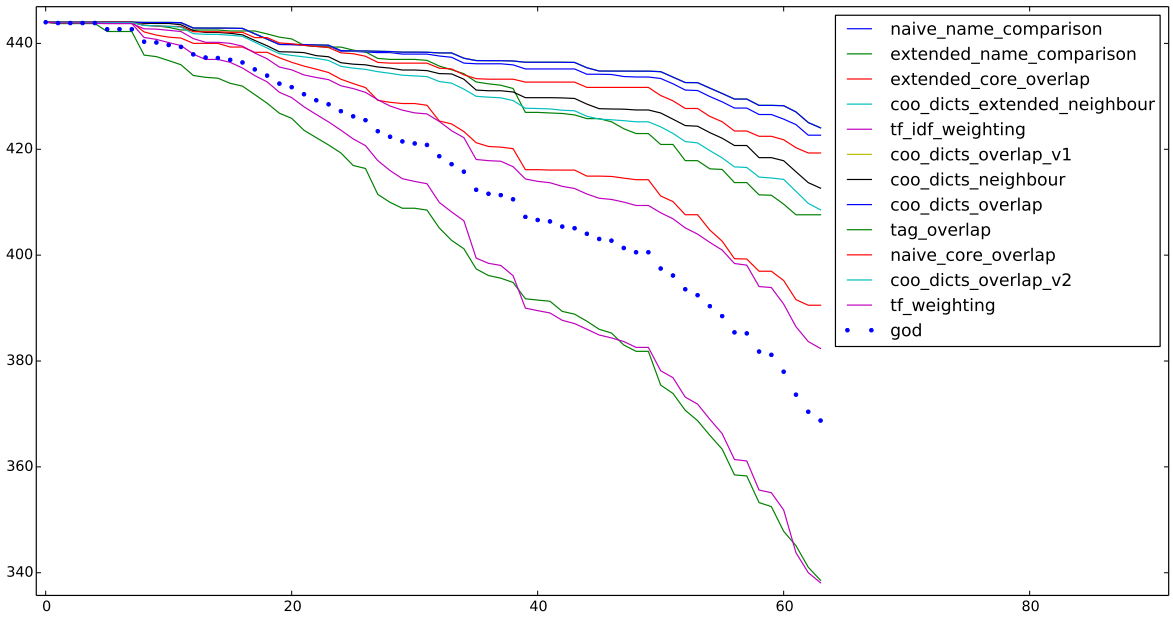

Figure 6: regrets

8.1.1 Conclusions

Summing up, tests of this kind were ran with the following parameters:

- Median, learning rates 0.2, 0.4, 0.5, 0.8, 1.
- Average, learning rates 0.4, 0.8, 1.
- Step by step average, learning rate 0.2.
- Median and average, learning rate 0.2, punish = True[5].

8.2 Test 2: feedback.

The second test was altogether different. Instead of emptying the sky and adding back clouds, we first make a full analysis, and then slowly give feed-

[5]This means that feedback was given not on known truths but also on unknown pairs (which, we remind, are assumed to have similarity 0'). More on this in the appendix.

back, monitoring the subsequent evolution of the belief state and the trustworthiness of the angels.

Part III
Appendices.

9 Appendix 1: punishing.

The decision to only give feedback on known truths (thus not penalising false positives [what I say is, but is not]) can be questioned. Also, no (negative) feedback is given on false negatives, and this is even more questionable.

In short, feedback is given to all and only those angels which have a non-zero opinion about a link which we know to be true. Angels which spot similarities where we know there are none (false positives), or angels who don't spot similarities where we know there are (false negatives), are not punished for this, for two different reasons.

Reasons for not punishing for false negatives: the main one I can see is that we chose to interpret the decision of the angels not as opinions about the relatedness of items, but about confidence ratios about such relation.

The difference is subtle but important: if an algorithm returns 0.5 when fed with two clouds, you can interpret that output in at least two ways:

relatedness "I believe these two items to be 0.5 related / they have 50% in common / they resemble to each other half of how they would if they were exactly equal."

confidence "I am not entirely sure, but 5 out of 10 these two items are related / I am 50% confident in the relatedness of these two clouds."

The most relevant outcome of this distinction is that on the relatedness reading, a 0/10 rating means 'my opinion is that these two items are not related', whereas on the confidence rating that would be a milder 'I have no reason to believe they are related, but I just don't know'.

Given that the algorithms we used were relatively naive and relied, as explained in the appropriate section of this report, on a (very) restricted subset of the available evidence, it seemed obvious to use the 'confidence' interpretation of their opinions instead of the other one: they have to be humble about their judgements. Their perceptual field is so narrow that an

output of 0 is hardly interpreted as a negative relatedness judgement, but rather as a 'here I can't see anything' statement.

As a consequence of these considerations, we chose not to punish the angels for their false negatives: if they can't see anything, it's not their fault and there's nothing they can do about it: it's not by lowering their (relative) trustworthiness that we will improve the situation.

Reasons for not punishing for false positives: First of all, the Knower's belief set is assumed to be 'all there is to know', literally, but in a dynamic frame such as Starfish's one, we want to always keep the discussion going. When the state of the discussion (the state of the art, if you wish) is not yet settled, when there is no agreement or just too little opinions to have a definitive decision, then the rationale of 'all there is to know' becomes questionable.

But most crucially, what matters for the purpose of classification is not the unrealistic target of all and only true items being detected as such (zero false negatives and false positives), but that the gap in the confidences is wide enough to allow for a nice separation of them from the false ones (via ranking, for suggestions, or by thresholding, for selection of links.)

And, as it comes out, the angels are already good enough to do this as they are. The average confidence in true links against false links is, as it has been previously shown, clearly in advantage of true links. So, instead of punishing an angel's relative trustworthiness regarding some link type because of its (many) errors in which he nonetheless has very low confidence, thereby lowering the future impact of his (few) right clues in which he has a higher confidence, we could just not give bad feedback on the low-confidence errors.

Clearly if an angel had high confidence on some or many errors, the situation would need more accurate consideration, and it might even be worth considering some thresholded triggering of feedback based precisely on a trustworthiness / confidence function, so that we give feedback on items only when the angel is trustworthy enough in his own suggestion.

A backing experiment. As a partial proof of the usefulness/correctness of this stance, we ran a test in which we set the punish flag to True, thus giving feedback to false positives (to give feedback on false negatives is currently just not possible in the framework, but it will in the future). Graphs of the outcome follows, and are rather self-explanatory.

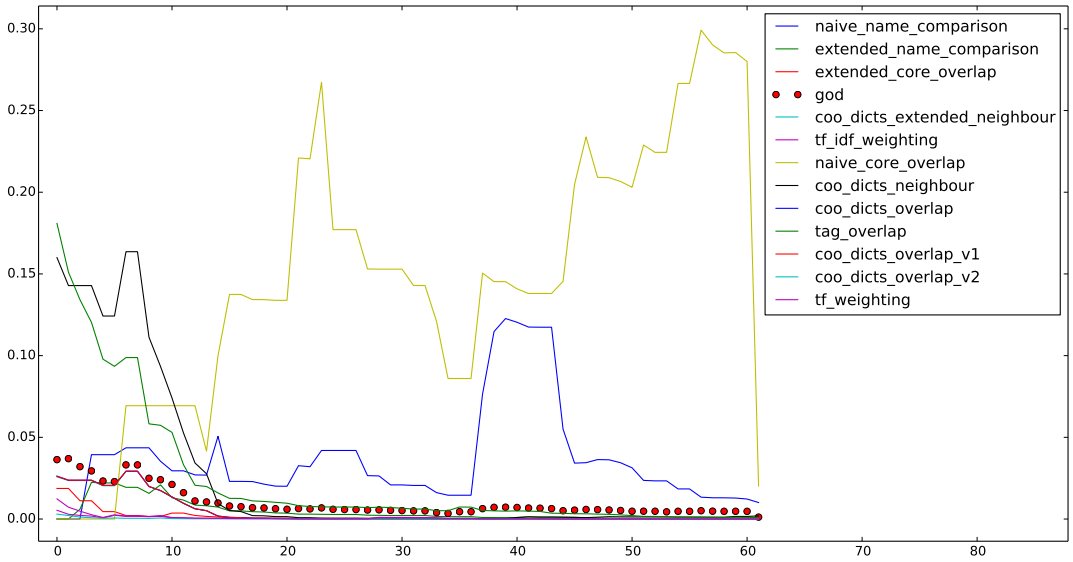

Figure 7: average belief in true links

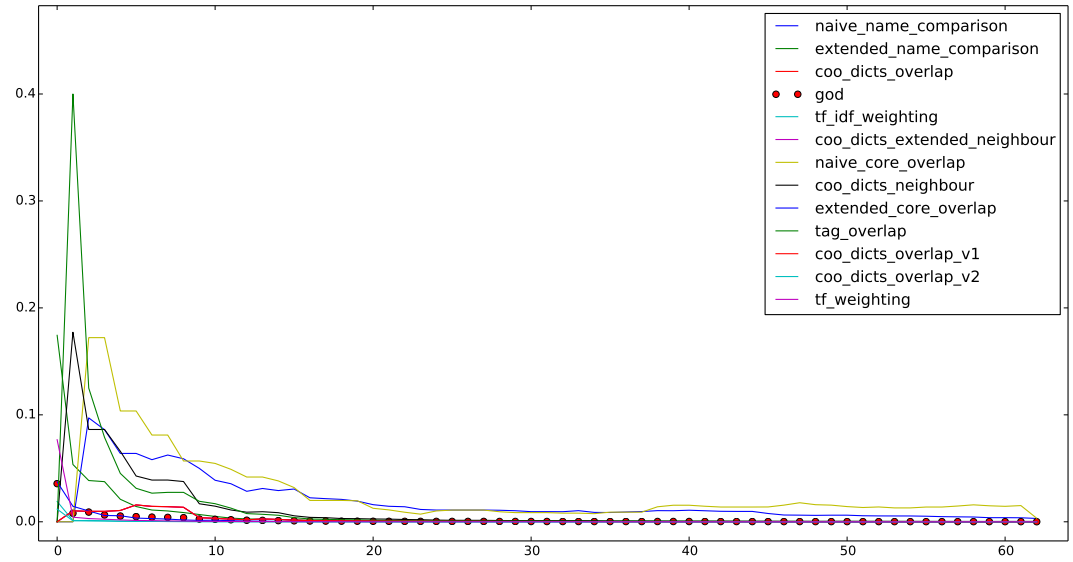

Figure 8: average belief in false links

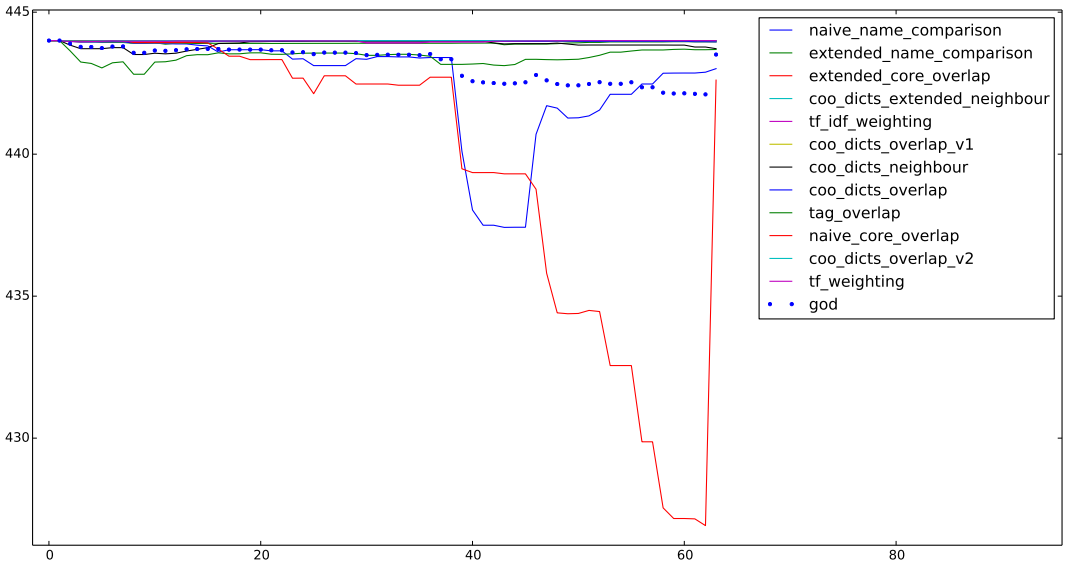

Figure 9: regrets

As before, the two following figures represent the outcome of a different test (on the present test): taking the weight sets as sampled during the evolution of god's beliefs as the present test (median, learning rate 0, punish) proceeded and assigning them to a different god, we have him refresh his belief set according to these weights and then plot the resulting regrets.

As you can see, also by comparing these figures to the previous ('healthy') ones, the regrets in this case touch the top (440) and seem stably hang to the asymptote. In the 'all links' version of the regret calculation, on the other hand, we have an incredibly descending curve which stabilises at a very low value (1250 for God's regrets), against the 2000 top in the non-punish case(s).

This is due to the fact that all trustworthiness ratios go down, and thus, being most links false (i.e. not in Starfish) the angel's guesses get statistically more correct as their weighted value approaches 0.

Of course we would like regrets to decrease (especially where God is concerned), but as you can see from the graph displaying the evolution of regrets on true links, this is not happening in an altogether desirable way.

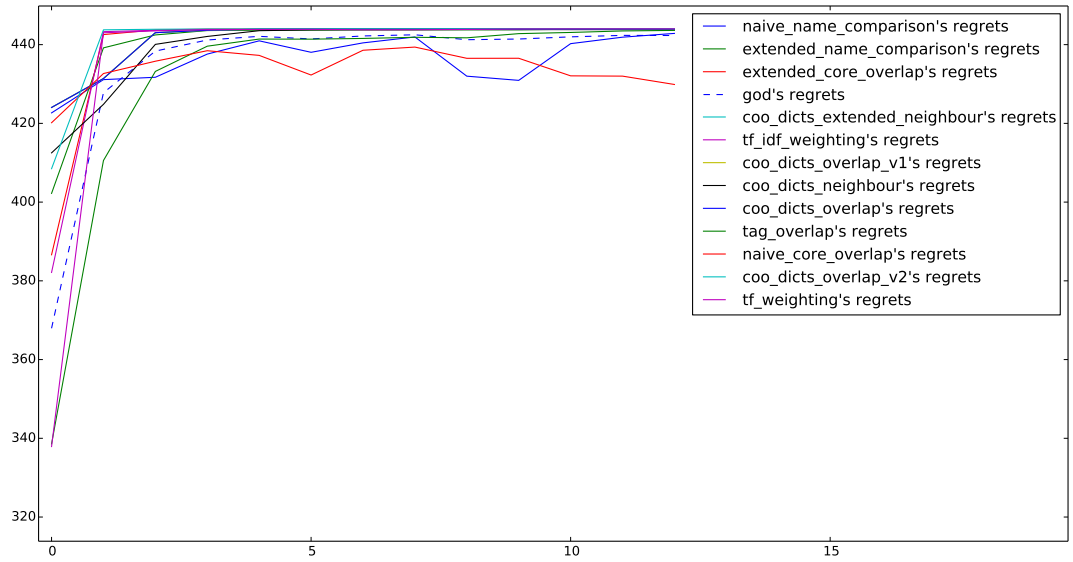

Figure 10: evolution of regrets (on true links)

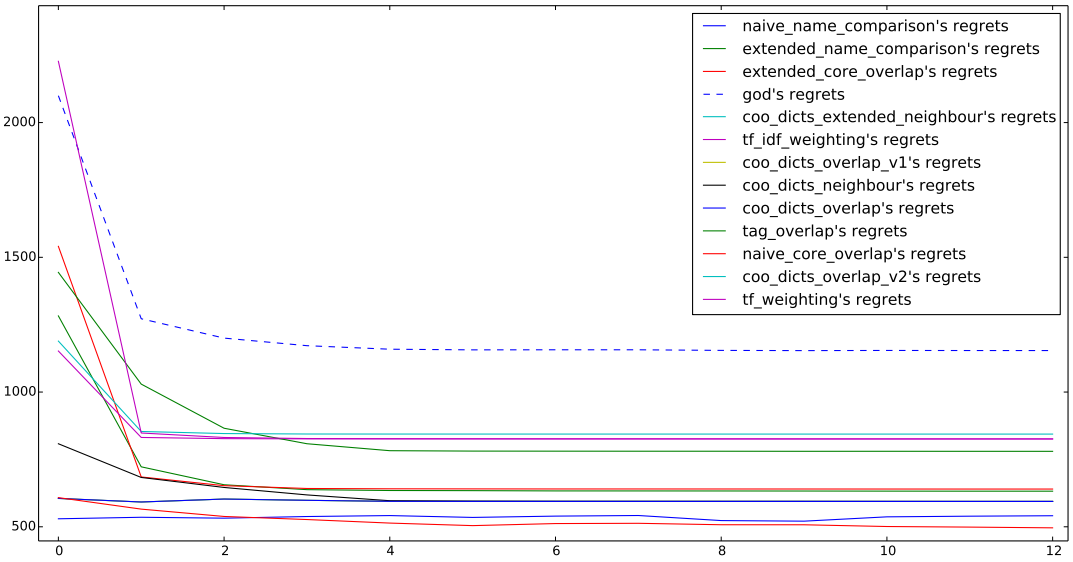

Figure 11: evolution of regrets (on all links)