Wednesday Sept 10 2014

VB[Base]->NN[true] finds verbs that take direct objects as far as I can see. It stops if it hits an IN.

we now have a list of verbs. We then take members of that group and find, for each verb, a list of nouns that follow that verb. We stop if it hits an IN before our noun. we now have lots of sets of nouns.

which of these groups overlap? what is an overlap? we can sort into well-overlapping groups. these are new groups. we can collect all the verbs that precede these nouns directly - without an IN intervening.

we can take any two POS labels a and b with closeness c. closeness may be a number of words of separation or the intervention of a third POS.

ASIDE need to add a way to, say, pair up with a word from a group but stop if any other member of the group comes first.

we then make a group of words in a that are followed by a word with label b within c. make that group a-w. we then take each of the a-w and make a group for each that is words labeled b that follow that specific word within c. call these b-w[]. again we can collapse some of these groups by finding groups that are close and coalescing.

we need criteria for groups that are useful. we now have a huge number.

ideas for rating groups as useful/interesting

1. size of the group

2. ok. big idea. take random sentences and mess them up. mix up the messed up sentences with original sentences and test your groups. if the group is useable at all and if it can distinguish between real and messed up sentences then it is useful.

a fully messed up sentence is a random jumble of words.

a partially messed up sentence switches only two words in the sentence either with each other or one word with a random outside word.

3. Take any feature of a sentence such as whether it contains some POS or not. you want the feature to be roughly present at 50% at least at first. look at the words in the sentence. look for groups that point to words in many of the sentences. use the presence or absence of groups as properties that create binary predictors of the required feature. create 100 to 1000 decision trees. evaluate the trees based on their ability to distinguish presence of requested feature in out-of-bag examples. for trees that are successful, give the concepts making up the nodes in the tree “usefulness” credits.

targets

language

* voice-to-speech, vts, lowering the probability of nonsense sentences relative to alternatives
* discussing plots, human causality networks
* intonation, stress, rhythmics and volume in language - understanding and generating.

games

chess

go

programming

learn from the open source universe

characterize the running of code using machine code

database

take any SQL and meander through it

list of data types

1. text sources. an array of modules loaded as needed
   1. word record: word, base word, pos
   2. sentence rec: array of word record for one sentence
   3. module: text source. an array or sentence recs
2. word map
   1. each unique word gets an entry. array indexed by the word
   2. word map entry. contains the following list of references
      1. data source. references to text source, module, sentence, word rec
      2. group
3. ancestor pattern
   1. list of pattern nodes. search activated once last node completes
4. group-specific finder pattern
   1. replace one or more specific node from ancestor pattern with one of its (the node’s) members
   2. contains list of members indexed by node
   3. ref to ancestor pattern
5. merger strategy
6. group
   1. indicator whether from finder pattern or merge strategy
   2. reference to group-specific finder pattern maybe valid
   3. reference to merge strategy. maybe valid
   4. array of member links
   5. member link. number of hits, number exist in docs reviewed after member created.
   6. sorted member links
7. maps to groups
   1. map group name to group.
   2. groups sorted by number of strong members

Do some supervised learning. Look for the two meanings of the word "that" . take a selection of sentences and mark which meaning of the word each represents . let the system learn a classifier. It should Use POS information as well as groups . Groups that are useful here get a big boost .

This may not be the latest version of this document.

Prioritize an idea mentioned earlier. Create two classes of examples sets. Choose a common word (word a) and then look for sentences that have a specific POS or member of a group within a specific distance (word b). (Perhaps use commas to affect distance calculation.) You want to apply two labels to the sentences – 0 if word b is missing (or too far away) and 1 if present. Now build classifiers to distinguish 0 from 1. As usual groups participating in successful classifiers are boosted.

Make a classifier a concept. If we can find a similarity metric between two classifiers, then we can have groups of classifiers.

Alternative ways forward:

A little stuck Nov 5 2014. Sentences with one error in them not identified by random forest using groups of words based on POS they are normally found in.

1. Switch to looking for patterns of POS groups broken up by punctuation and CC

2. Just switch Anchor away from that. Keep Score on good anchors.

3. switch to a system without an anchor. Use distances between positions. Manage the n^2 by selecting 2 or 3 from n. At first use punctuation/CC to divide off the relevant section. Later, learn the divide offs.

4. play with the rf. Learn to limit the depth of the tree. Justification: in language at least the relations can’t be so complex as to justify so many nodes of the tree. Don’t forget min\_size

5. abandon rf. Go back to groups only. Don’t just use single word membership. See if the pattern find algorithms hold for the test pattern

6. move away from POS groups. All patterns, including the errors are pretty likely. Get more specific groups

It, for, an, as

Jan 14, 2015

1. Pick a dep relation
2. Find a group that relates a single source word to elements of the group with that relation
3. Find another group with a different source word but with a different relation
4. Make a new group from words in one group that also exist in the other

Alternative

Find a set of groups (src groups), criteria:

A specific finder

A specific GrpParam value (say the

Create a group of words that appear in all or some fraction of the src groups

Group creation:

Find a number of candidate groups. Criteria:

A specific finder

A specific GrpParam value (say the dep relation stored in 0)

Order candidates by number of elements and select a number of the good ones

Fill (does not use sentence rec):

Find groups that have some overlap of elements

Sometimes I have a dep with me, sometimes I don’t. Can we figure out when? Suggested path:

1. Pick a word.
2. Create groups for all deps with that word
3. Find all examples of it in a number of modules
4. Choose a dep relation from (2.) that appears in some but not in other examples
5. Create a dataset where each record is a sentence and each field is 1 or zero on the other deps, 1 if that dep is present.
6. Run RF on dataset

There will be cases where if a dep appears it almost always appears and cases where it sometimes appears. What have I learned from this? I can tell whether a sentence is structured right for that word. So this suggests whether the word is out of place as in voice recognition.

How many other words would predict similarly? If very few have I found a peer? Can this be used to recognize synonyms?

\*

How do we prepare swap samples? We are trying to grow an RF that will determine which of two alternatives for a word is the correct one. (Say VR is not sure whether you said apples or addles). Assume the rest of the words in the sentence we are 100% sure about. We can create samples of sentences with each word, remove the missing word and learn which, for a given sentence, the more likely choice between the two is. Presumably a sentence which was supposed to have word a in, looks very different from a sentence with word b in.

So we pass a sentence to Stanford NLP twice. Once with word a and once with word b. We then analyze the dep relations that the nlp outputs.

We can analyze these relations using either:

1. Seeing how likely each relation is. Likely means whether we have other examples of that relation. (In our example, we might have “sweet red apples” and “sweet red addles”. The NLP might decide that the word sweet is an adjective, but we have almost no adjectives at all for addles. We need the concept of high surprisingness (Number of occurrences divided by rank in word count, unexpectedness, opposite of expectedness) for an element of a group. We also need the concept of strength/surprisingness for the group. This means that a highly used word should have a large group of elements with lots of occurrences (maybe the total number of occurrences of all elements). This is a strong group. If it has a weak group, this is surprising. Therefore a verb will have a weak group for adjectives. Thus having an adjective on addles creates a negative signal. Even if the alternate word was also a noun, for most cases, the word sweet would be a positive signal for apples and not for other nouns. This option simply adds up the signals to determine the more likely option.
2. Build a dataset out of the relations. Use an RF to decide which of the two words is the right one, given the sentences. However, first we need to grow an RF. We take sample sentences. Alternatives:
   1. Take only samples from sentences with the word a. Replace a with b and put both through NLP. Classify with RF. Problem. How would the RF know that adjectives are bad for addles? It sees adjectives in both un-replaced and replaced examples. It doesn’t know that this distinguishes.
   2. Take samples of sentences with *“a”* and mix with sentences with b but label them correctly. The RF should see the lack of adjectives in b cases. Problem. Relies on NLP to call sweet an adjective. Maybe it will call it an adverb and then RF will classify an “a”-sentence as “b”. Potential solution. If we check sweet in the adverb groups we won’t set a strong signal which would translate to a strong negative signal. We can enter a high negative number in the dataset. The question is, which adverb groups do we use? Do we use the dep-based groups? The groups for a? Perhaps this is a test for peer words. We check the groups of peer words and put their signals into the dataset.

I want to get a better concept of strength for an element in the group. Take that element’s word and look up its word count. Find a set of words with similar word count whose average is the same as the seed word. If the count for the seed word is much higher than the others, its strength is higher. This would work well for a simple group. However, multiple conditions might require calculating strength relative to elements that are in the group used to make the new group.

Group strength can be calculated as described above. Compare the seed itself to others with the same word count. Compare the total Number Occurs of the groups, if it is high, the relation is likely.

\*

Dep count can be increased significantly by making collapsed deps. Take any dep group that has elements with very large counts and create a group of complex-deps that bridge gov and dep across that element. This should be doable recursively but current methods of creating NodeListItem require explicit unrolling of multiple levels of complexity. So we would need a 2-level complex dep builder, a 3-level etc.

\*

Can we learn by creating patterns out of the primeval noise? Just start looking for any non-uniformity in the random data of NLP. For example, take sentences. Do some sentences have one dep and others not? The goal is to create low-level structures that can be used in higher levels of learning.

\*

There is an analogy of the form:

A is to B as C is to B

And there is

A is to A as C is to D

1. MASON : CARPENTER :: STONE : WOOD  
   or  
   MASON : STONE :: CARPENTER : WOOD

1. SQUARE : FOUR :: HEXAGON : SIX  
   or  
   SQUARE : HEXAGON :: FOUR : SIX

1. COPPER : OXYGEN :: METAL : NON-METAL  
   or  
   COPPER : METAL :: OXYGEN : NON-METAL

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mason | Is-a | Job | That-uses | Stone |
|  |  |  |  |  |
| Carpenter | Is-a | Job | That-uses | Wood |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mason |  |  |  | Carpenter |
| v |  |  |  |  |
| Stone |  |  |  | Wood |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | job | material |  |  |
| uses | Mason | Stone |  |  |
| uses | Carpenter | Wood |  |  |

Conclusion of the game.

1. Find a relation that ties appears at least twice, tying operand 1 and operand 2 for two or more different tuples of operands
2. Find a category label for each set of operands.

So if Wood is missing from the foursome, we can fill it in. The real world application for such a test is another relation other than “uses” in this example. For example, “deliver” Stone to the Mason, we see that Stone and Mason are related by “uses” so we can learn that you “deliver” Wood to the Carpenter. However, this example doesn’t really need the categories. It would use the categories if “uses” has lots of operands that are not part of these categories. In that case the inference to deliver Wood would be stronger

Look at it another way. We have a group Mason and Carpenter attached to seed Job. Stone is in another group seeded by Material, but Wood is NOT. If Stone is found in a group seeded by Mason, and Wood to be found in another group seeded by Carpenter, we can expect that Wood should also be seeded by Material. We need a sense of the frequency of Wood relative to Stone because if Wood is far more common than Stone and does not appear then we should perhaps conclude that it does not belong in the group after all. Essentially we are looking to fill in gaps caused by lack of examples or input data. If the number of examples should already be providing the data – and not by analogy – then we should be relying on that.

\*

Links:

<http://olst.ling.umontreal.ca/pdf/Claveau-LHomme-tke05.pdf>

<https://en.wikipedia.org/wiki/Lexical_function>

<https://en.wikipedia.org/wiki/Collocation>

<http://www.eleto.gr/download/Conferences/5th%20Conference/5th_24-11-Schmitz%20Klaus-Dirk_Paper.pdf>

<https://en.wikipedia.org/wiki/Semantic_similarity>

<https://en.wikipedia.org/wiki/Analogy>

HU AI Center

<http://mlai.cs.huji.ac.il/index.php>

<http://groups.csail.mit.edu/genesis/papers/StoryWhitePaper.pdf>

<https://www.youtube.com/watch?v=1-ep_QqYVvQ&list=PLT-roSWIpp1Gs60ZULB6TwTaUk4X1-I76&index=3>

<http://allenai.org/content/publications/ValenzuelaHaMeaningfulCitations.pdf>

http://allenai.org/papers.html

\*

We measured surprising-ness by looking at the other members of the same group that have more or less the same word count. So far the experiment was run with the deps of the “det” group. These are words such as a, the, some, which. There are only 26 of these. They are fairly well spaced out in the count-frequency listing. The groups for each word (chest, child, city) were quite small. The surprising-ness was normally 100%. Not much achieved here.

However, what is interesting is that some of these words have “a” in their group only, some have “the” and some have “these”. That clearly distinguishes these words and we should be able to categorize into large groupings along these lines. You could call that the group surprising-ness. Perhaps, regular groups can be built along these lines.

Expanding on the idea at the end of the last para. Take a group with a very high total count but a small number of members. The group ancestor should be a primary one such as AllDepGovs, AllDepDeps or AllPosWords. Take the relation defining the group and divide the group of words on the other side of the relation according to which words have which. For example the det dep group consists of 26 elements. For each, make a group of the det govs. There will now be a division of what is basically nouns into 26 groups.

We need to create an unexpectedness quotient for this division. Say in the “these” group which should only include plural nouns, there is the word “child” with only one occurrence. Our current algo should succeed for this.

\*

An idea for how to separate two different meanings of the same word A. Find a similar word B. Collect sentences of B and grow an RF to distinguish it from a broken version. Take a collection of A sentences and classify them into valid and invalid using B’s RF. Grow an RF to learn this A by B classification. The more predictable the more valid the idea that we have a homonym (or homograph) on our hands.

\*

Can we use RF to build a distributed representation of a word? We need the idea of a word that, by usage count should be part of a group but is not. Say A and B appear strongly in a number of groups but A is \*surprisingly\* absent from some of B’s groups. In this case, usage statistics would dictate that A’s absence is surprising. We can build a vector out of the groups A does and “definitely” does not belong to.

\*

Surprisingness did not do very well in the last attempt. At least it did not seem to. Here’s another try. Take a look at the average (or some such) of the elements of the group. If a word has a very large NumOccurs despite a usage count far below the average, it is surprisingly present. This itself may not be very useful. What may be more useful is how surprising it is that a word is not an element or that it has a surprisingly low *NumOccurs*. If a word’s usage count is as high as or higher than the mean, yet has a low NumOccurs, it is surprisingly absent. Of course, most words will not be in a group and that will not be interesting. However, if, say A is related to B yet is not in the group, that IS interesting.

We need a better concept of average. It can’t just be an average of the usage counts. It could be NumOccurs / Usage. However, that does not work so well as it favors the low usage counts. Perhaps Monte Carlo can be used here, but unclear to me how.

\*

A word could be represented by its vector of membership in all groups. We might be able to do the linear vector algebra trick of deep learning here. However, the dimensions are not normalized. We would need to use context tests to normalize the dimensions. Another idea is to use the concept of group degeneracy. If two groups give the same score on the context test then maybe they are degenerate. Another test is if they have almost the same members.

\*

Short term plan:

* Make a longer run of 3Way. Use any popular pairs and not just nsubj and dobj
* Create a function that can pull out pattern groups of an ancestor group that have a large number of strong members. Perhaps combine with print
* Look for groups NOT having an overlap. Ideally they have one or two group parameters that are the same and nevertheless particularly DON’T overlap.
* Create an index for words showing which groups the word is in
* Build similarity of words
* Add some spare floats for groups so that we don’t have to recreate for every variable added.
* Build the context test for deps
* Add different groups from high similarity words in the context test. Assign scores to helpful groups.

\*

<https://colah.github.io/posts/2014-07-NLP-RNNs-Representations/>

torch

<http://torch5.sourceforge.net/manual/newbieTutorial.html>

<http://en.wikipedia.org/wiki/Torch_%28machine_learning%29>

<https://en.wikipedia.org/wiki/Argument_from_analogy>

\*

Build your own corpus. Break sentences so that only childishly simple sentences are left. Pick a small vocabulary of very common words. Keep only sentences that have ONLY the words from the vocabulary. Use this corpus to learn right from wrong.

\*

Wildly successful experiment. What does it mean?

1. Created dep nears (pairs) from 100 popular words. So we have groups seeded by a word and a dep relation governors that contains a group of dependents for that relation and that seed word.
2. Take another set of popular (30 offset) words and see which of these it belongs to. Then find other words that belong to the same group. We used in all these only strong (> 5) num occurrences. A good match has between two words means that they both belong to many such groups.
3. Take the best matches, pair by pair. For each group the first of the pair belongs to strongly, print out (a) the average occurs to count ratio of the group (GOPC) and the ratios for each of the pair (OPC / GOPC).

We get a printout like:

Printing comparison between she(77813) and we(67789)

For group DepNears[%0%dobj][%1%said] with average OPC 0.0105598, and she has O 13 and OPCR 0.015821 while we has O 1 and OPCR 0.00139696

For group DepNears[%0%nsubj][%1%\_] with average OPC 0.0152278, and she has O 16 and OPCR 0.013503 while we has O 13 and OPCR 0.0125935

For group DepNears[%0%nsubj][%1%comes] with average OPC 0.000111932, and she has O 8 and OPCR 0.918506 while we has O 0 and OPCR 0

For group DepNears[%0%nsubj][%1%could] with average OPC 0.000166326, and she has O 25 and OPCR 1.93164 while we has O 8 and OPCR 0.709528

For group DepNears[%0%nsubj][%1%discovered] with average OPC 0.000118764, and she has O 8 and OPCR 0.865672 while we has O 7 and OPCR 0.869469

For group DepNears[%0%nsubj][%1%making] with average OPC 5.26122e-005, and she has O 6 and OPCR 1.46559 while we has O 3 and OPCR 0.841154

For group DepNears[%0%nsubj][%1%pulled] with average OPC 6.07514e-005, and she has O 9 and OPCR 1.90386 while we has O 0 and OPCR 0

For group DepNears[%0%nsubj][%1%rose] with average OPC 0.000669436, and she has O 6 and OPCR 0.115183 while we has O 3 and OPCR 0.0661079

For group DepNears[%0%nsubj][%1%says] with average OPC 0.0145351, and she has O 42 and OPCR 0.0371346 while we has O 1 and OPCR 0.0010149

For group DepNears[%0%nsubj][%1%smiled] with average OPC 0.00384786, and she has O 18 and OPCR 0.0601175 while we has O 0 and OPCR 0

For group DepNears[%0%nsubj][%1%stood] with average OPC 0.00110092, and she has O 31 and OPCR 0.361871 while we has O 5 and OPCR 0.066997

For group DepNears[%0%nsubj][%1%turned] with average OPC 0.00384207, and she has O 53 and OPCR 0.17728 while we has O 5 and OPCR 0.0191976

For group DepNears[%0%nsubj][%1%will] with average OPC 0.00142341, and she has O 5 and OPCR 0.0451426 while we has O 11 and OPCR 0.113999

For group DepNears[%0%pobj][%1%as] with average OPC 0.00543873, and she has O 7 and OPCR 0.0165405 while we has O 12 and OPCR 0.032548

There are a couple of interesting points. (1) She and we definitely belong together. In most cases they have similar OPCRs. However, see that the OPCR for “we” “comes” is zero. That’s because you don’t say “we comes”. In that sense *we* does not belong. It also seems unusual to say “we smiled” even though it’s allowed. *She* and *he* does better. However, *his* and *her* are good for most cases but *her* is also the direct object of *she* while *his* is not the direct object of *he*.

What do we do with this?

1. We can simply create groups out of these matching pairs – regardless of subtleties.
2. Once we make such groups we can take all groups seeded by these pairs and combine them. This makes bigger groups which will mean more elements of those groups crossing the strong threshold.
3. We can use analogy to promote less popular words. Say A and B are match-pairs but A is far more popular than B. A belongs to group 5 but B’s count is below group 5’s average. So B would be there if we had a larger corpus. So, put B in anyway. This is not true if B’s count is high enough to get in, but nevertheless has 0 occurences. See next point.
4. We can break down individual words into sub-meanings. Word A is like B in most senses but clearly not in others. A relates through dep *d* to 1, 3, 5, 6 while B relates to 1, 3, and 5 but not 6. Thus if a word C is like B in *every* way, we should not always draw an analogy to A. Also 1, 3, and 5 must belong in a single group, related, but not in every way, to 6. “her” is a good example. It is the possessive female but also the direct object female pronoun. These two are sub-meanings of “her”. Does that mean that “her” (poss) is a heteronym of “her” (dobj)?

\*

These analogies look good, but what do we do with words that are not so frequent. We can combine words such that the groups they seed also combine. This can give us many more occurrences. However, we are counting groups that a word belongs to. Now the same seed combines multiple times in many different ways with other seeds producing many groups. However, the same source group will contribute to multiple combined groups. Therefore belonging to multiple groups could be meaningless because it is caused by belonging to the source group.

This suggests creating combined groups by using the group file ids of the source group. Say group A has match-pair with B, C, & D. We can create 3 + 3 + 1 groups AB AC AD ABC ABD ACD and ABCD. Since you don’t just combine the elements, you can tell when you are repeating sources. Yes, but which combine should I use?

So far don’t know what to do with this direction yet. Certainly keep the sources and refer to them in combining rather than lose the data in a mush

\*

Trying to produce a list of good looking analogies. That should allow exploration of how to make good analogies and use them well.

A shares a few groups with B. (Say A is “their” and B her) However, B clearly is missing in group [dobj-gov/like] (you can like her but you can’t like their).

A is Paris and B is Rome. A and B have a lot in common. However B is missing from in-France. You check and A is missing from in-Italy. One strategy is to jump on any group that A is missing from. However that will give errors like Paris:in-France::Rome:near-the-sea. Or Paris:has-Eiffel-tower::Rome:dry-summers.

Looks like you ideally want:

1. To establish that A and B are related by finding a few groups that they are in with high OPCR (Occurrence per Count Ratio to OPC of the group OPC/GOPC)
2. Find a group A is in for which A has a high OPCR and B is zero or very low (one-sided group)
3. Find as many one-sided groups as possible; one set for each side
4. Relate groups. A pair of one-sided groups, one from each side, strongly connected in other ways, is a great analogy

However, this is only one kind of analogy. Others:

1. Paris:in-France::Rome:in-Italy
2. Lives-in:Bardstown::Lives-in:Paris. (Bardstown is a small town in Kentucky). Why is this important? Because there may be only one reference to what happens in Bardstown in 50,000 books and millions for Paris. You can learn that you can drink coffee in Bardstown. However, you don’t want to learn that there is an Eiffel tower in Bardstown. In order to avoid this, you have a few options. (1) Assume that B (Bardstown) is a very little guy in a group of a few popular guys. We can see what groups they share. Ignore those they don’t share. Whatever they share applies to B. (2) Assume there is only one peer of significant popularity in the one group B belongs to, P (Paris). Find P’s peers. Find groups that are not particular to P. Make the leap that B should be a member of P’s peers and apply all the properties to B too. Make this a little more stringent by applying only the properties that apply to all the members of P’s peers to P.
3. Disanalogy: Consider the group: He, she, they, we, I (personal pronouns - PP) as opposed to “it” (impersonal pronoun - IP). The PP can be the subject of eat, learn, loved, etc. The IP is not. (Well, not impossible but below the radar statistically). The IP can be the nsubj of “true” whereas only “they” out of all PPs came close. We learn that IP is like PP but is different in other ways, this holds even though we cannot find a specific connection between the one-sided groups as in 1 above.

\*

Analogy looks very promising, but getting the configuration parameters right will be HARD. Two words are related if they are in the same group but the potential factors are: number of elements in group, log(num els), total num occurs, log of, how do you discount Disanalogy, how do you factor in the popularity of the words, how do you deal with low digits when they might be flukes?

The solution may be dependent on the task. Perhaps we have to use RF to learn these parameters. Of course this means the learning depends on the nature of the test cases.

\*

Changing tactics a bit. Being related is based on the total number of occurrences in in the same groups divided by the total number of hits. Counter-analogies are the ones that will consider surprise and stuff like that.

\*

Latest series of thoughts on analogy. How do we relate two words? Let’s create a spectrum from -1 to 1 for each group. We can even try adding squares as if we’re talking distance in multiple orthogonal dimensions. If one is clearly in a group you get 1. If the other is too, then no distance. If zero in a group you get -1 and if the other is too, then, again, no distance. If one is in a group and the other doesn’t have the hit-count to get in, then no distance is attributed. This means that we might have divide by the number of groups you’re in, or a very low hit-count

\*

Simple way to handle large/small hit count issue. Whenever you want a result biased towards the more popular words, multiply the score by the NumOccurs or hit count. For example, you’re trying to find close words and only get unusual words in your “closest” list. Use this trick then.

\*

May have an explanation for the effect observed that the overage OPC for very popular words is always lower than unusual words. Perhaps, for any group there are more intense members and less intense members along a continuous spectrum. Less intense members have lower OPC. For high popularity words, even less intense members make it into the group. For low popularity words, only the high intensity words get in. Therefore the average OPC of the popular words is lower. QED

\*

Distance. When two words are in the same small group, their distance is small. When two words are in a large group their distance may be very small but might be quite large. Now assume that there is a group G that A is in but not B. If G is small distance A->B is at least medium. If G is large, distance is large.

If two words both have OPC above GOPC for the group, they are in the same group. If they both have OPC’s but below GOPC, score = (OPC(A) / GOPC) \* (OPC(B) / GOPC). Max each ratio at 1. If both are zero, both are outside. If one is above a threshold and the other is zero, we have a disparity proportional to OPC / GOPC maxed at 1.

To determine if OPC is zero or very low because of low count, calculate k / GOPC where k is a small number such as 3 or 5. If the count is below that, we do not have a zero.

\*

Let’s create two concepts, closeness and distance. Instead of treating them on the same spectrum, we add each up separately.

\*

C:\Users\Eli\Downloads\FAIA263-0627.pdf

From analogical proportions in lattices to proportional analogies in formal concepts.

<http://afflatus.ucd.ie/Papers/analogy2014.pdf>

Analogy as an Organizational Principle in the

Construction of Large Knowledge-Bases

Tony Veale, Guofu Li

Abstract

A capacity for analogy is an excellent acid test for the quality of a

knowledge-base. A good knowledge-base should be balanced and coherent, so that

its high-level gene

C:\Users\Eli\Downloads\2022-8469-1-PB.pdf

Reducing the Dimensionality of Data Streams

Using Common Sense

Catherine Havasi, Jason Alonso, and Robert Speer

<http://web.media.mit.edu/~lieber/Publications/AnalogySpace-AAAI.pdf>

AnalogySpace: Reducing the Dimensionality of Common Sense Knowledge

<https://wordnet.princeton.edu/man/wndb.5WN.html#sect10>

wordnet file structure

<http://ai.stanford.edu/~rion/swn/>

enhanced wordnet?

<https://hal.inria.fr/hal-01095344/document>

Applying belief revision to case-based reasoning

Julien Cojan and Jean Lieber

<https://en.wikipedia.org/wiki/Formal_concept_analysis>

<http://www.cl.cam.ac.uk/~jrh13/papers/ab.pdf>

comparison of automated theorem provers and computational algebra systems

A short survey of automated reasoning

John Harrison

<http://ccc.inaoep.mx/~ariel/2013/2013%20Training%20inter-relatedclassifiersforautomaticimageclassification.pdf>

grouping image classes using properties they have in common

Training inter-related classifiers for automatic image classification

and annotation

Peixiang Dong, Kuizhi Mei, Nanning Zheng, Hao Lei, Jianping Fan

<https://en.wikipedia.org/wiki/Open_Mind_Common_Sense>

<http://homepages.inf.ed.ac.uk/ddiochno/research/publications/CommonNet.pdf>

<http://wordnet.princeton.edu/wordnet/download/current-version/>

<https://en.wikipedia.org/wiki/WordNet>

<https://en.wikipedia.org/wiki/Cyc>

C:\Users\Eli\Downloads\FAIA263-0627.pdf

Analogy and Formal Logic

C:\Users\Eli\Downloads\2022-8469-1-PB.pdf

Reducing the Dimensionality of Data Streams

Using Common Sense

Catherine Havasi, Jason Alonso, and Robert Speer

Explanation of AnalogySpace and an optimization

<http://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm>

<http://en.wikipedia.org/wiki/Newton%27s_method_in_optimization>