The Gavagai Living Lexicon

Magnus Sahlgren†, Amaru Cuba Gyllensten†, Fredrik Espinoza†, Ola Hamfors†, Jussi Karlgren†
Fredrik Olsson†, Per Persson†, Akshay Viswanathan† and Anders Holst⋆
†Gavagai, Slussplan 9, 111 30 Stockholm, Sweden
⋆SICS, Box 1263, 164 29 Kista, Sweden
[firstname.lastname]@gavagai.se, aho@sics.se

Abstract

This paper presents the Gavagai Living Lexicon, which is an online distributional semantic model currently available in 20 different languages. We describe the underlying distributional semantic model, and how we have solved some of the challenges in applying such a model to large amounts of streaming data. We also describe the architecture of our implementation, and discuss how we deal with continuous quality assurance of the lexicon.

Keywords: Distributional semantics, word embeddings, lexical resources

1. Introduction

The availability of large amounts of text data in open sources like online news and social media, coupled with the recent developments of scalable and effective techniques for computational semantics, opens up new possibilities for lexicographic research and resource development. As a complement to traditional lexica and thesauri, it is now possible to automatically build lexical resources by mining large amounts of text data using unsupervised machine learning methods.

The perhaps most well-known example of a data-driven thesaurus is the Sketch Engine,[1] which features distributional thesauri for some 60 different languages (Kilgarriff et al., 2014). Another example is the `polyglot` Python library,[2] which contains word embeddings trained on Wikipedia for 137 different languages (Al-Rfou et al., 2013). There are also several academic projects that provide data-driven thesauri, like the Wortschatz project,[3] and the JoBimText project.[4]

This paper presents a *continuously learning* distributional thesaurus — the **Gavagai Living Lexicon** — that updates its semantic model according to the input data stream in an online fashion, and produces lexical entries in several different languages.[5] The lexicon currently includes several different types of entries: string similar terms, topically related terms, sequentially related terms, multiword terms, and semantically similar terms. In the following sections, we describe how we use an online distributional semantic model to identify and compile these various relations. We also describe the architecture of our implementation, and discuss how we deal with continuous quality assurance.

[1] `www.sketchengine.co.uk`

[2] `pypi.python.org/pypi/polyglot`

[3] `corpora.informatik.uni-leipzig.de`

[4] `maggie.lt.informatik.tu-darmstadt.de/jobimtext`

[5] As of 2016-03-10, the lexicon includes the following 20 languages: Czech, Danish, Dutch, English, Estonian, Finnish, French, German, Hebrew, Hungarian, Italian, Latvian, Lithuanian, Norwegian, Polish, Portuguese, Romanian, Russian, Spanish, Swedish.

2. (Online) Distributional Semantics

Distributional Semantic Models (DSMs) represent terms as distributional vectors that record (some function of) co-occurrence counts collected within a context window surrounding each occurrence of the terms. There are many different approaches to building such vectors, ranging from simple accumulation of co-occurrence frequencies to more advanced methods based on matrix factorization (Österlund et al., 2015) and artificial neural networks (Turian et al., 2010). The particular framework we use for building DSMs is called Random Indexing (RI) (Kanerva et al., 2000; Kanerva, 2009), which provides an attractive processing model for streaming environments because it can be formulated as an online model that processes data as soon as it becomes available. This is done in RI by accumulating distributional vectors incrementally by vector addition; every time a term a occurs in the text, its distributional vector $\vec{v}(a)$ is updated according to:

$$\vec{v}(a) \leftarrow \vec{v}(a_i) + \sum_{j=-c, j\neq 0}^{c} w(x^{(i+j)})\pi^j \vec{r}(x^{(i+j)}) \qquad (1)$$

where c is the extension to the left and right of the context window surrounding term a, $w(b)$ is a weight function that quantifies the importance of context term b (the default setting is $w(b) = 1$ for all terms), $\vec{r}_d(b)$ is a *random index vector* that acts as a fingerprint of context term b, and π^j is a permutation that either rotates the random index vectors according to the position j (what we call *order-coded* windows) or the direction (what we call *directional* windows) of the context items within the context windows, thus enabling the model to take word order into account (Sahlgren et al., 2008).

Distributional vectors accumulated with Equation (1) encode semantic properties, and can be used to quantify semantic similarities between terms. RI can also be used to accumulate the equivalent of a topic model (Steyvers and Griffiths, 2006), which can be used to quantify topical relations between terms. Such an RI topic model can be formulated as:

$$\vec{v}(a) \leftarrow \vec{v}(a) + \sum_{t \in T} w(a,t)\vec{r}(t) \quad (2)$$

where T is the set of documents in the data, $w(a,t)$ is the weight of term a in document t, and $\vec{r}(t)$ is the random index vector of document t.

RI handles data and vocabulary growth by using fixed-dimensional vectors of dimensionality d (such that $d \ll (V,T)$ where V is the size of the vocabulary and T is the cardinality of the document set; d is normally on the order of thousands), which means that a new context does not affect the dimensionality of the distributional vectors. We simply assign a new d-dimensional random index vector to each new context; the dimensionality of the distributional vectors remains constant regardless of the size of the data.

3. Online Frequency Weighting

The most important step when building a DSM is dealing with the skewness of the frequency distribution. This amounts to reducing the impact of high-frequent items, since they pollute the distributional representations. For the topic vectors, we can simply use some version of TFIDF, such as:

$$w(a,t) = \log \mathrm{TF}(a,t) \times \log \frac{T}{\mathrm{DF}(a)} \quad (3)$$

where $\mathrm{TF}(a,t)$ is the frequency of term a in document t, and $\mathrm{DF}(a)$ is the document frequency of term a. Note that we take the log of the TF, which is done in order to control for topicality (i.e. we effectively require that terms occur several times in documents in order to count as relevant).

For the semantic vectors, frequency weighting is normally done by either using stop lists (based on either part of speech tags or frequency thresholds), or by using association measures such as Pointwise Mutual Information (PMI).[6] The online setting complicates matters when it comes to frequency weighting, since we have to operate with a continuous stream of data. There are streaming versions of PMI (Durme and Lall, 2009), but since we use RI to accumulate distributional statistics in a *distributed* representation, we do not have access to individual co-occurrence counts that can be transformed with PMI.

The way we approach frequency weighting in the online setting is to weight each individual vector addition by an online frequency weight $w(b)$ that operates only on the accumulated frequency $f(b)$ of the current context item b and the total number of unique context items seen thus far V (i.e. the current size of the growing vocabulary):

$$w(b) = e^{-\lambda \cdot \frac{f(b)}{V}} \quad (4)$$

where λ is an integer that controls the aggressiveness of the frequency weight.

This weighting formula returns a weight that ranges between close to 0 for very frequent terms, and close to 1 for not so very frequent terms. Figure 1 shows the distribution of weights plotted against the frequency of the 20,000 most frequent terms in the ukWaC data.[7] As can be seen in the figure, the weighting formula gives the desired effect of reducing the impact of high-frequent terms, while giving a nearly constant weight to medium- and low-frequent items. Note that the use of the weighting formula does not completely remove all co-occurrences with high-frequent terms; they have a weight *close to* zero, but not exactly zero, and at the beginning of processing they will still have a useful weight, since the weighting formula has not yet learned the frequency distribution.

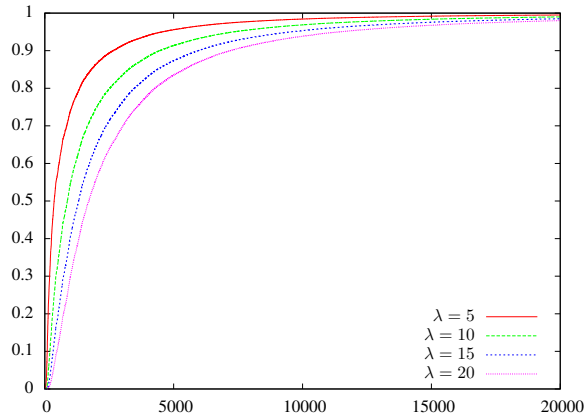

Figure 1: Distribution of weights with different λ for the 20,000 most frequent terms in the ukWaC.

4. Dominant Eigenvector Removal

Equations (1) to (4) describe an online DSM that accumulates fixed-dimensional distributional vectors from a continuous stream of data. However, an artifact of only using addition for accumulating the distributional vectors in RI is that all vectors will tend towards the dominant eigenvector of the distributional space (this is the case also for standard DSMs built without RI). One way to reduce the gravitational force of the dominant eigenvector without having to compute a full principal component analysis is to utilize the fact that RI can be seen as the first step in a *power iteration*, which extracts eigenvectors by iterating: $v' \leftarrow \frac{Av}{||Av||}$ (i.e. multiplying the vector v with the matrix A, and normalizing). Thus, by summing all distributional vectors (and normalizing the result), we are effectively approximating the dominant eigenvector of the distributional space. We can then remove this dominant vector from all distributional vectors by using:

$$a' = a - \frac{a \cdot b}{|b|^2} \quad (5)$$

where a is the original vector and b is the vector we want to remove. This operation is known as the *Gram-Schmidt process* (Golub and Van Loan, 1996), which has also been referred to as *orthogonal negation* (Widdows, 2003).

[6] $\mathrm{PMI}(a,b) = \log \frac{p(a,b)}{p(a)p(b)}$, where $p(a,b)$ simplifies to the co-occurrence count of a and b (normally multiplied with the number of tokens in the data), and a and b simplifies to the frequencies of a and b.

[7] `wacky.sslmit.unibo.it`

5. Collocation Cleaning

Since the semantic distributional vectors are built from co-occurrences, they will encode a significant amount of collocations. This unfortunately has a negative effect on the distributional model, since the collocations affect the result of nearest neighbor searches by introducing neighbors that share the trace of the collocate (as an example, the term "white" may occur significantly in the collocation "white house," which can introduce nearest neighbors that only share the co-occurrence with "house," like "brick" and "beach"). Such collocations can be identified by looking for *spiky patterns* in the distributional vectors. As an example, consider the following two vectors:

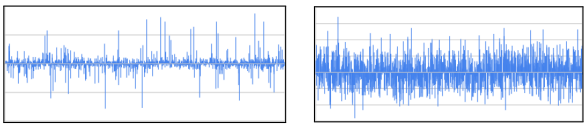

Figure 2: Example of distributional vectors produced with RI, one with a spiky pattern (to the left), and one without (to the right).

The vector to the left has a very spiky pattern, with a small number of elements having much higher values than the rest, while the elements in the vector to the right have much more even values. The elements with high values in the vector to the left come from one very frequent collocation, and we can identify that collocation by taking only the elements with high values (i.e. the spiky pattern) and searching (using inverse permutations) for a random index vector that looks like the spiky pattern. The term whose random index vector is most similar to the spiky pattern is the most likely collocate, and since we have used permutations to accumulate the distributional vectors, we also know at what position in the context window the term has occurred. This means that we can now form a multiword unit by concatenating the two terms in the correct order, and we can then include that multiword unit as a term in its own right in the distributional model. Note that this method is also able to detect skip-grams like "ministry ... defence." We use the following condition for identifying spiky vectors:

$$0.01 * d > N_{|v_i| > \frac{\max |v_i|}{2}} \tag{6}$$

i.e. if the number of elements N that have values that are higher than a threshold (defined as the highest value in the vector divided by 2) is less than 1% of the dimensionality. If a distributional vector satisfies this criterion, we make a *collocation vector* consisting of *only* the spiky pattern (i.e. all elements satisfying the right-hand part of Equation (6)), and we can now use that collocation vector to remove the impact of the collocation on the distributional vector. We do this by *weighted* orthogonalization:

$$a' = a - \delta \cdot \frac{a \cdot b}{|b|^2} \tag{7}$$

where a is the original distributional vector, b is the collocation vector, and δ ranges between 0 and 1. Note that with $\delta = 1$, Equation (7) equals the Gram-Schmidt process in Equation (5). We use $\delta < 1$, since we want to retain some traces of the collocation in the distributional vector (even though "house" might primarily co-occur with "white" in the collocation "white house," it is possible that there are other occurrences of "white house" that are *not* instances of the collocation in question).

By continuously running a collocation extraction job, the online DSM is able to incrementally identify multiword terms of higher order; starting with bigrams, and continuing with higher-order n-grams – in theory all the way up to frequently recurring phrases (idioms) and even whole sentences.

6. Lexicon Compilation

The dominant eigenvector removal, the collocation cleaning, and the multiword identification are performed as preprocessing steps before using the distributional vectors for lexicon compilation, which is done by extracting the k nearest neighbors to the n most frequent terms in the vocabulary (what we call the *target vocabulary*). We use $k = 50$ and n is on the order of 100,000 to 200,000 depending on the amount of incoming data for the language in question. We also extract the 10 most salient left and right neighbors to each term in the target vocabulary by doing nearest neighbor search over the random index vectors using inversely permuted distributional vectors, as described in Sahlgren et al. (2008). Multiword terms are identified using the collocation criterion, and for the semantic and topical nearest neighbors, we also compute the relative neighborhood graph (Cuba Gyllensten and Sahlgren, 2015), which uncovers the different usages of terms and can be seen as a form of word-sense discovery. We also include string similar terms in the lexicon, which is computed using standard Levenshtein distance (Levenshtein, 1966). In order to avoid information overload in the GUI, we use the following heuristic for the semantic neighbors: if the neighbors are grouped into more than 5 relative neighborhoods, we only show the top 5 neighborhoods; if not, we show all 50 nearest semantic neighbors. For the left and right neighbors, we only show neighbors with a similarity that exceeds a predefined threshold.

7. System Architecture

The Gavagai Living Lexicon is implemented using Enterprise JavaBeans, and is designed to work with large and streaming data in a decentralized and scalable fashion. The distributional vectors are stored in a cluster of Redis instances,[8] and nearest neighbor compilation is performed on a dedicated GPU server. The GPU solution is advantageous to using, e.g., a Hadoop cluster, which is optimized for small calculations that are repeated on large amounts of input data, while the nearest neighbor compilation requires large (and very regular) numerical calculations on fairly small amounts of input data, a scenario that GPUs are ideal for. The GPU server carries out approximately 250 million point-wise comparisons per second. The result of the nearest neighbor calculation is stored in MySQL, which serves as the back-end for the Living Lexicon. The

[8] `redis.io`

Living Lexicon can be accessed either through an API,[9] or using a web-based GUI, shown in Figure 3.[10]

8. Data

The Living Lexicon is continuously fed with data from a range of different sources, including news media, blogs, and forums. The input data is retrieved from a number of different data providers, e.g., Trendiction[11], Twingly[12], and Gnip[13], as well as by means of internal processes. The data is normalized to a unified internal format, a process that includes language detection, assignment of source definitions, de-duplication, and time-stamping, before the texts are subject to processing with the purpose of updating the lexicon. The data flow contains millions of documents each day; at peak periods, the flow can reach some 20 million documents each day, which amounts to more than a billion tokens each day.

The lexicon is currently available in 20 different languages. The amount of data differs considerably between languages: English is by far the largest language, followed by Russian, Swedish, German and French.

9. Examples

Table 1 provides a few illustrative examples of the type of entries learned by the lexicon. For each entry in the lexicon, the table lists all types of relations currently represented in the lexicon: string similar terms identified using Levenshtein distance; commonly preceding and succeeding terms computed using inverse permutations; the nearest semantic neighbors, sorted according to their relative neighborhood graph; multiword units identified using the collocation criterion; and topical neighbors, sorted according to their relative neighborhood graph.

Note that the **preceding** and **succeeding** columns contain different numbers of terms, which is an effect of using a global threshold for the syntagmatic similarities. The entry for "great" illustrates the somewhat noisy nature of the syntagmatic relations; the succeeding terms all make sense, but the preceding terms seem less intelligible; the syntagmatic relations for "great" would probably have benefited from using a slightly more aggressive threshold. Note also that the topical neighbors do not always make sense. This is due to the lack of topically coherent discussions in online media featuring the terms in question. Out of the four examples in Table 1, "apple" has the clearest topical neighbors, which indicates that this term is used in topically more coherent contexts than the other terms in this example. For terms like "great" and "what the hell", which occur in topically very diverse contexts in online media, the topical neighbors seem more or less random.

The term "what the hell" does not have any string similar terms, and it has not been part of any multiword units (or rather, the system has *thus far* not detected any multiword units featuring "what the hell" as a part, but that might happen in the future). For the relative neighborhood graphs, which can be seen as a representation of the different usages of terms (and hence as a sort of word-sense discovery), the lexicon also provides a best guess for labels that describe the various usages. These labels (which are not shown in the table, but are available in the web-based GUI) are produced by looking at common preceding or succeeding terms among the members of each relative neighborhood.

10. Continuous Quality Assurance

As should be obvious from the examples provided in the last section, the quality of the lexicon entries can differ considerably. Since one of the points of the Living Lexicon is to investigate how an unsupervised DSM will develop as it continuously reads uncontrolled data from online sources, such qualitative differences are to be expected; the lexicon is nothing more than a representation of the current state of online language use. However, this makes it slightly challenging to perform quality assurance and evaluation of the lexicon.

One way we tackle this problem is to include the Living Lexicon in our automated testing routine, which monitors our continuous deployment process, with system changes pushed to production on a daily basis. These automated tests track and identify what the impact of system development is on the quality and consistency of the lexicon. The automated tests include both standard benchmark tests

[9] developer.gavagai.se
[10] lexicon.gavagai.se
[11] trendiction.com
[12] twingly.com
[13] gnip.com

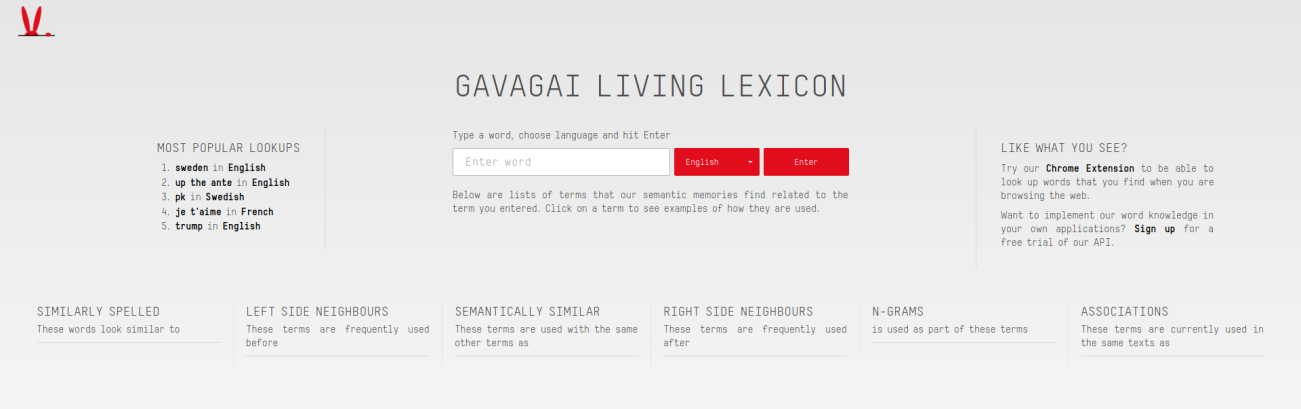

Figure 3: The Gavagai Living Lexicon GUI.

Entry	String	Preceding	Semantic	Succeeding	Multiwords	Topical
suit	suity	cabbies	[suits	the needs	the suit	[how to eat]
	suitx	navy	law suit]		follow suit	[law suit
	suita	strongest	[class-action lawsuit		followed suit	rebelled
	suits	civil	the suit		civil suit	legions]
	suite		class-action suit		bathing suit	[suits
	sunit		wrongful death lawsuit		filed suit	auto racing]
	subit		statement of claim		black suit	[clavin klein
	sumit		class action suit		law suit	euphoria
apple	applea		[the apple	ceo tim cook	the apple	[apple's
	applen		apple's	pie	apple inc	apple watch
	applet		apple inc's]	juice	apple pay	the apple watch]
	apples		[android wear		apple tv	[ceo tim cook
	appl		jailbroken		apple iphone	tim cook]
	appel		android-based		apple store	[the iphone
	appletv		apple ios		the apple watch	the iphone 6s
	appleid		iphone 4s		apple pie	rumours
great	greate	brick walls	[terrific	lakes	the great	[jurgen
	greats	special interests	brillliant]	lengths	great way	unconfirmed reports]
	grenat	manly	[tremendous	pride	great idea	[bookmarked
	grea	pragmatic	phenomenal	running start	great britain	thanks for sharing
	greta	polluting	unbelievable	hbo series	great day	blogroll
	gret		even greater	britain	great lakes	websites
	greater		immense	business acumen	great things	obliged]
	greatly		incredible		great value	[don't move
what the hell		wondering	[what the fuck			[wincing
		wonder	wtf			meandering
		figured	dafuq]			17-point]
			[what on earth			[revulsion
			what the fuss			inhabiting]
			who the heck			[run with it]
			what ever happened			[take me home]
			what went wrong			[constructors]

Table 1: Example entries in the Gavagai Living Lexicon.

(such as those mentioned in the next section), as well as more specific in-house test sets, where we measure lexicon agreement with external resources. These tests are focused exclusively on the semantically similar terms, since the other types of relations in the lexicon are either more volatile in nature (e.g. the topically related terms), or lack standardized benchmarks (e.g. the multiword terms).

Continuous automated quality assurance of a learning semantic model is admittedly a difficult problem, and we do not pretend to have a definite solution in place.[14] We would like to avoid including humans in the loop as far as possible to facilitate continuous delivery, but we acknowledge that human plausibility ratings might be the most relevant evaluation for this type of resource.

11. Batch Evaluation

We also regularly perform batch experiments and compare the underlying DSMs to other state-of-the-art algorithms. As an example, we use three different semantic tests: the TOEFL synonym test,[15] the BLESS test (Baroni and Lenci, 2011),[16] and the SimLex-999 similarity test (Hill et al., 2014). As data, we use a dump of Wikipedia, which is tokenized by removing punctuation, and down-casing all terms. The resulting data contains some 1.1 billion tokens. We compare the online model defined in Equations (1), (4), (5), (6) and (7) with two other well-known types of models: a standard DSM with terms as context in a ± 2-sized window using the raw co-occurrence counts (DSM), optimized frequency thresholds (Frequency), and standard PMI weighting (PMI), and the two models included in the `word2vec` library,[17] SGNS and CBoW, both with optimized parameters.[18] For the online models, we use 3,000-dimensional vectors and a window size of ± 2. The results are summarized in Table 2.

As can be seen from the results in Table 2, the proposed

[14] Evaluating dynamic behavior of semantic models is a challenge in itself (Karlgren et al., 2015).

[15] Kindly provided by the late Professor Thomas Landauer.

[16] The BLESS test is originally formulated as a test where models compute similarities for 200 target terms to terms representing 8 different kinds of semantic relations. We only use the *co-hyponymy* relation in our experiments, and require that the model ranks one of the terms from the co-hyponymy relation higher than the terms included in all the other relations in the test data.

[17] `code.google.com/archive/p/word2vec`

[18] Both models perform best in our experiments using 500-dimensional vectors and a context window of 2.

	TOEFL	SimLex	BLESS
DSM	63.75	0.09	69.34
Frequency	77.5	0.28	70.85
PMI	81.25	**0.38**	69.34
SGNS	81.25	0.34	80.5
CBoW	85	0.35	**84.5**
Order-w	81.25	0.28	72
Order-w-e	81.25	0.29	72.5
Order-w-e-c	78.75	0.32	79
Dir-w	85	0.29	70
Dir-w-e	86.25	0.30	71
Dir-w-e-c	**88.75**	0.33	77.5

Table 2: Results on three different semantic tests using three variations of a standard DSM, the two models in the `word2vec` library, and two different versions of the online distributional method: using order-coded context windows (Ord-), and using directional context windows (Dir-), both with online frequency weights (-w), eigenvector removal (-e), and collocation cleaning (-c).

online model is competitive in comparison with both standard DSMs and neural network-based models. The online model produces the best score on the TOEFL synonyms (88.75%), while the results on the SimLex-999 test for the online model is comparable to the SGNS and CBoW models (0.33 vs. 0.34 and 0.35) but lower than the PMI model, which produces the best result on SimLex-999 (0.38). For the BLESS test, the CBoW model produces the best score (84.5%), with the online model in second place (79%), and the PMI model only performing on a par with the raw frequency matrix (69.34%). These results demonstrate that the online model produces results that are comparable to, and in some cases even exceed, the current state of the art, and that both the proposed online model and the models included in `word2vec` produce consistently good results over the three different tests used in these experiments.

One weakness of the evaluation measures used here is that, at least in theory, a model could produce good results on the tests, while still returning irrelevant terms as nearest neighbors. An optimal test for a DSM would therefore evaluate the quality of the nearest neighbors directly, but this is a difficult problem, since these nearest neighbors will be highly data-dependent. We are not aware of any such test suites. In order to get an idea about the quality of the nearest neighborhoods in the different models, Table 3 provides some examples of the five nearest neighbors for four different terms in the online model, the CBoW model, and the PMI model. These examples indicate that all three models produce reasonable neighborhoods, and that the online model and the CBoW model are the most similar in this respect. The PMI model is the only model that produce collocates as neighbors (e.g. "very" and "luck" as neighbors to "good"), while both the online model and the CBoW models exclusively feature paradigmatically similar terms as neighbors.

Since the online model and the CBoW models produce both very similar neighborhoods and similar results on the three different test suites, it may be informative to look at how

Dir-w-e-c	CBoW	PMI
suit		
lawsuit	suits	suits
suits	lawsuit	filed
countersuit	lawsuits	lawsuit
complaint	countersuit	wearing
counterclaim	complaint	jacket
play		
playing	compete	plays
compete	playing	playing
participate	perform	played
perform	plays	season
play	participate	players
good		
bad	bad	excellent
excellent	decent	luck
poor	excellent	decent
decent	poor	poor
better	mediocre	very
blue		
purple	purple	yellow
yellow	red	red
turquoise	yellow	purple
lightblue	bloodred	pink
reddishbrown	skyblue	green

Table 3: Examples of the five nearest neighbors to "suit," "play," "good," and "blue" in the online model, the CBoW model and the PMI model.

the different models behave when exposed to increasing amounts of data. Figure 4 shows learning curves for the different models; the left-hand plot shows the results for the TOEFL test using increasing amounts of Wikipedia data, the middle plot shows the results for the BLESS test, and the right-hand plot shows the results for the SimLex-999 test. Note the log scale of the x-axis in the plots, which makes it easier to see the development of the models at the beginning of processing, which is arguably the most critical phase. As can be seen from these learning curves, the PMI model consistently outperforms the other models when there is only limited data available, but when data increases, the results for the PMI model seem to reach a plateau fairly quickly, while the results for the other models continue to improve. For the SimLex-999 test, the results for the PMI model even decrease after having seen approximately half of the Wikipedia data (the top result for the PMI model on SimLex-999 is 0.39).

12. Conclusion

This paper has presented an implementation of an online distributional semantic model that learns various types of lexical relations between terms from a continuous large stream of text data. The model is used to produce an unsupervised distributional thesaurus – the Gavagai Living Lexicon – that contains entries for hundreds of thousands of terms in several different languages. The motivation for developing the lexicon is threefold: firstly, the lexicon is a

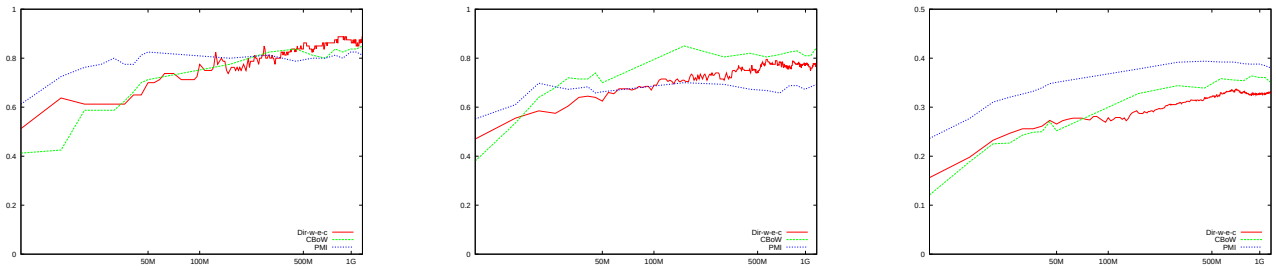

Figure 4: Learning curves for the TOEFL, BLESS, and SimLex-999 tests for the three different models (RI-dir-w-e-c, CBoW and PMI) using Wikipedia as data. Note the log scale of the x-axis.

valuable resource for higher-level text analysis systems and natural language processing applications, since it provides a constantly up-to-date semantic base layer; novel terminology and novel language use is automatically included and updated in the lexicon continuously, without the need for editorial intervention. Secondly, the lexicon is valuable from a scientific perspective, since it constitutes a long-term experiment that investigates how a DSM develops when continuously fed with large amounts of uncontrolled data from online sources. Thirdly, the lexicon is a useful tool for lexicographic research, since it represents current language use in online media; by looking at the entry for some term over time, we can get a good understanding of how the use of the term changes and evolves.

In this paper, we have described the underlying online DSM, and the various processing steps we use to compile the lexical entries. We have also briefly discussed the system architecture, and the input data, and we have exemplified the varying quality of entries in the Living Lexicon. We have argued that such qualitative variance is to be expected, since the online DSM only reflects the current language use on the internet; extremely diverse contextual behavior limits the statistical regularities that can be utilized by the DSM. Since quality assurance of an unsupervised continuously learning semantic model trained on uncontrolled online data is a challenging problem, we have also provided results from batch experiments using controlled data sets, which demonstrate that the online DSM used by the lexicon performs on a par with current state-of-the-art algorithms.

We conclude that the Living Lexicon demonstrates the viability of utilizing unsupervised techniques for compiling lexical resources, and we hypothesize that we will see a much wider use for such systems in the near future.

13. Bibliographical References

Al-Rfou, R., Perozzi, B., and Skiena, S. (2013). Polyglot: Distributed word representations for multilingual nlp. In *Proceedings of CoNLL*, pages 183–192.

Baroni, M. and Lenci, A. (2011). How we blessed distributional semantic evaluation. In *Proceedings of GEMS*, pages 1–10.

Cuba Gyllensten, A. and Sahlgren, M. (2015). Navigating the semantic horizon using relative neighborhood graphs. In *Proceedings of EMNLP*, pages 2451–2460.

Durme, B. V. and Lall, A. (2009). Streaming pointwise mutual information. In *Proceedings of NIPS*, pages 1892–1900.

Golub, G. H. and Van Loan, C. F. (1996). *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA.

Hill, F., Reichart, R., and Korhonen, A. (2014). Simlex-999: Evaluating semantic models with (genuine) similarity estimation. `http://arxiv.org/abs/1408.3456`.

Kanerva, P., Kristofersson, J., and Holst, A. (2000). Random indexing of text samples for latent semantic analysis. In *Proceedings of CogSci*, page 1036.

Kanerva, P. (2009). Hyperdimensional computing. *Cognitive Computation*, 1(2):139–159.

Karlgren, J., Callin, J., Collins-Thompson, K., Gyllensten, A. C., Ekgren, A., Jürgens, D., Korhonen, A., Olsson, F., Sahlgren, M., and Schütze, H. (2015). Evaluating learning language representations. In *Proceedings of CLEF*. Springer.

Kilgarriff, A., Baisa, V., Bušta, J., Jakubíček, M., Kovář, V., Michelfeit, J., Rychlý, P., and Suchomel, V. (2014). The sketch engine: ten years on. *Lexicography*, 1(1):7–36.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics—Doklady 10, 707710. Translated from Doklady Akademii Nauk SSSR*, pages 845–848.

Österlund, A., Ödling, D., and Sahlgren, M. (2015). Factorization of latent variables in distributional semantic models. In *Proceedings of EMNLP*, pages 227–231.

Sahlgren, M., Holst, A., and Kanerva, P. (2008). Permutations as a means to encode order in word space. In *Proceedings of CogSci*, pages 1300–1305.

Steyvers, M. and Griffiths, T. (2006). Probabilistic topic models. In T. Landauer, et al., editors, *Latent Semantic Analysis: A Road to Meaning*. Laurence Erlbaum.

Turian, J., Ratinov, L., and Bengio, Y. (2010). Word representations: A simple and general method for semi-supervised learning. In *Proceedings of ACL*, pages 384–394.

Widdows, D. (2003). Orthogonal negation in vector spaces for modelling word-meanings and document retrieval. In *Proceedings of ACL*, pages 136–143.