Lucene, MetaMap, and Language Modeling: OHSU at CLEF eHealth 2013

Steven Bedrick and Golnar Sheikshabbafghi

Center for Spoken Language Understanding, Oregon Health & Science University, Portland, OR, USA
{bedricks,sheiksh}@ohsu.edu

Abstract. The Oregon Health & Science University team's participation in task #3 ("addressing patients' medical questions") of this year's eHealth CLEF campaign included submissions from two different retrieval systems. The first was a traditional, Lucene-based system modified from one used in previous years' TREC-med campaigns; the second was a novel system that used statistical language modeling techniques to perform text retrieval. Since 2013 was the first year of our participation in this campaign, our focus was on familiarizing ourselves with working on a corpus of web text, as well as putting together a proof-of-concept implementation of a language-model retrieval system. We submitted three runs in total; one from the novel system, and two from our Lucene-based system, one of which made use of the National Library of Medicine's MetaMap tool to perform query expansion. In general, our runs did not perform particularly well, although there were several topics for which our language model-based retrieval system produced the best P@10. Future work will focus on pre-indexing text normalization as well as a more sophisticated approach to query parsing.
Keywords: Lucene, MetaMap, language model, skip-grams

1 Introduction

Most research into medical information retrieval can be categorized into one of two broad and fuzzy groupings. The first ("type A") concerns itself with the information needs of clinicians, and focuses on searching specialized databases in response to specific and well-informed topics. The second category ("type B") deals with so-called "consumer" information needs: searches conducted by non-medical users (often patients or family members) over the Internet. Earlier medical IR evaluation campaigns have tended to focus on the first of these two groupings.

For example, for several years TREC included a medical track in which participants built and tested search systems designed to index and query electronic health records in order to identify patients matching particular textual descriptions (e.g., "Patients admitted to the hospital with end-stage chronic disease who are offered hospice care")[1]. Without in any way diminishing the substantial difficulties inherent in performing high-quality open-class IR over a clinical database, we believe that it is safe to say that this family of medical IR system enjoys several significant advantages over the aforementioned second family of medical IR system (systems designed for the lay population, and intended to work on a general web corpus).

A system designed for indexing and querying a clinical data repository of some kind has the advantage of a (relatively speaking) predictable data schema, as well as (again, relatively) a finite amount of content- both in terms of amount (i.e., how many unique records) as well as kind. After all, while there may be a very large number of types of pathology report, that number is generally both knowable and tractable. A search system designed for consumers and the open Web, however, must be able to handle an essentially infinite variety of input documents and user queries, and must do so with far less context (about both its corpus as well as its users[1]) than do its more constrained cousins.

The third task of this year's ShARe/CLEF eHealth track features just such a search scenario. The task is described in full elsewhere[3]; as such, we will keep our description brief. The task was an open-query document search over a corpus consisting of ≈1.6 million web pages ostensibly containing health-related information. The topics consisted of quasi-natural-language phrases (e.g. "is there a connection between multiple sclerosis and dysplasia in oesophagus") that represented the sorts of queries that actual patients might enter into a search engine such as Google.

Our group submitted runs derived from two different retrieval systems (described in section 2). While our results were— with one or two exceptions— not particularly impressive (see Section 3), we feel that we have laid a solid technical foundation for next year's CLEF campaign. Furthermore, our experience highlight several important differences between "type A" and "type B" medical retrieval systems in terms of how best to use external resources.

2 Methods

We submitted runs from two separate systems. The first system was s very traditional IR system based on the Apache Lucene[2] open-source toolkit. It was essentially a spiritual successor to the system used by the OHSU team for the 2011 and 2012 TREC medical tracks[4]. The second system was a novel system that uses techniques from statistical language modeling to perform retrieval.[3] The two systems were quite different in terms of their operation, and we will describe each in turn.

[1] While the information literacy of clinicians certainly varies[2], it is safe to say that this variability is smaller than that found among the users of a general-purpose medical search engine.

[2] http://lucene.apache.org

[3] See Chapter 12 of Manning, et al.'s "Introduction to Information Retrieval" for an overview of the general approach[5].

2.1 Traditional System

The first of our two systems followed a very traditional architecture for a text retrieval system, in that it featured a standard inverted index paired with vector-space retrieval model. As mentioned above, we used the Lucene open-source IR toolkit (version 4.2.1) to build the system, with no major modifications from its out-of-the-box configuration. Our intent was to develop a system to use as a reasonable baseline, and Lucene let us accomplish this with minimal difficulty. Lucene also provided us with robust index creation tools as well as a rich query language.

This baseline Lucene system used the default Lucene `StandardAnalyzer` query processor. Our system had an alternative query processing mode, which we used for one of our additional runs. This mode makes use of our existing MetaMap-based query parser used in previous years' TREC campaigns (see [4]), described in greater detail below.

Indexing Due to the large size of the corpus, building the Lucene index of the documents proved to be a non-trivial task. We chose to use a 500-node Hadoop[4] cluster to facilitate the process. Hadoop is an open-source implementation of the Map/Reduce pattern, which was first popularized by Google as a way of easily parallelizing certain computing tasks. A complete description of Map/Reduce is beyond the scope of this document (see Dean & Ghemawat[6] for a more thorough description); in short, a Map/Reduce program splits a task into two steps, "*map*" and "*reduce*." In the *map* step, each input document is processed in parallel and is transformed into a set of key/value pairs $\{< k_i, v_i >, < k_j, v_j >, ...\}$. In the *reduce* step, the key/value pairs are aggregated by their key into sets of the form $\{< k_i, \{v_{i_1}, v_{i_2}, ...\} >, < k_j, \{v_{j_1}, v_{j_2}, ...\} >\}$, and each key's associated values are processed together. The ultimate point of this approach is that each execution of the *map* step can be run completely independently from any others, and a similar amount of parallelization can often be achieved in the *reduce* phase.

The canonical example is a distributed word-counting operation. In this case, the mapper would take as input a document, and emit for each token in the document a key/value pair in which the key is the token and the value is the number of times that that token appeared in that document. The reducer, then, would take as its input a single token along with a set of counts (one from each input document that contained that token); its job would be to sum the counts and emit the total number of occurrences of that token in the corpus. Because each document can be counted independently of the others, and each token's occurrence counts can be summed independently of any other tokens' counts, a word-counting program using this approach can benefit greatly from a parallel computing environment.

The Map/Reduce model lends itself extremely well to the creation of inverted indices. Consider the most trivial case, in which the *map* step emits terms and

[4] http://hadoop.apache.org

postings as keys and values, and the *reduce* step produces posting lists. For our purposes, we used Map/Reduce to produce a Lucene index. Each mapper processed a subset of the entire corpus, and produced a single index shard; the shards were then run through the Lucene API's index-merging tools to produce a single, large index. The final index size was approximately 7.5 gigabytes, which proved to be well within Lucene's capabilities. We indexed both the document titles as well as their bodies, after stripping the bodies of their HTML tags.

MetaMap As mentioned above, our baseline system uses Lucene's default `StandardAnalyzer` to process free-text queries. During previous years' TREC campaigns, we developed a query parser that uses the National Library of Medicine's (NLM) MetaMap tool[7] tool to attempt to identify query terms that are "medically-relevant." MetaMap uses a variety of NLP techniques to map unstructured text to concepts from the Unified Medical Language System (UMLS) Metathesaurus[8].

Our query parser has a variety of operational modes, including several that perform query expansion by including sibling entry terms for any concepts matched from the NLM's Medical Subject Headings (MeSH) indexing system. For a complete description of the operation of this part of our system, consult our 2012 TREC Medical Track working notes paper[4]. In short, our query parser takes unstructured text as input, and in an unsupervised manner produces a (sometimes complex, and often suboptimal) query in Lucene's syntax, making use of various Boolean operators as appropriate.

In the simplest operation mode, our query parser uses MetaMap to analyze the free-text queries and identify any biomedical concepts. From there, we use the UMLS to identify possible synonyms for these concepts, and link those together using Boolean "OR." Each term group is then linked using Boolean "AND." The parser contains several "stop-word" lists— sets of CUIs or UMLS Semantic Types that it ignore, and not include in its final queries. Typically, CUIs or Semantic Types end up on this list due to being overly common in either queries or in documents (e.g., "Patients").

this approach worked reasonably well when used in previous years' campaigns, which involved querying electronic medical records for specific patient profiles. We were unsure as to how well it would work when querying the less formally-written and similarly less-focused content found in the present task, but decided that it was worth trying.

2.2 Language Model System

In addition to our traditional baseline system, we submitted a run from an entirely novel retrieval system. This system used techniques borrowed from statistical language modeling to attempt to identify documents that were statistically similar to the queries. Conceptually, we are using the language model to tell us, for each document, the probability of that document generating the query. More "relevant" documents should, in principle, have a higher probability than less "relevant" documents.

Slightly more formally, and using notation from [9], our approach computes a language model M_d for each document in the corpus d. Then, for any given query q, we use the language models to compute $P(q|M_d)$ for each document d, and can rank the documents in descending order of this probability.

This approach presents us with some interesting experimental possibilities. There are a wide variety of different text normalization approaches, model smoothing techniques, scoring functions, etc. that we wish to experiment with (see Section 4).

Text Normalization In any language modeling task, the decision of how to normalize one's input is critical. We were somewhat aggressive in our normalization approach. We first dropped all non-ASCII letters, since the task *a priori* only involved English-language queries over English-language documents. After alphabetical pruning, we compiled a list of more than 3 million tokens, most of which were not actual words but were instead fragments, numbers, etc. etc.

We then computed document frequencies for each token in order to compile a list of "stop tokens." We were primarily concerned with removing overly common tokens. By manually examining the relative frequencies of tokens, we set a document frequency threshold of 700,000— that is, we considered tokens appearing in more than that many documents to be "too common," and excluded them from both the documents as well as the queries.

Language Model The language modeling approach we used was very similar to a standard bigram language model with an absolute discounting backoff scheme.[5] However, instead of using strict bigrams, we instead counted tokens that co-occurred within an 11-word window (i.e., word pairs that had a maximum distance of ten words). In other words, if two tokens occurred relatively *near* to each other, they were counted as a bigram. Furthermore, the distance calculation took place *after* pruning stop words, so the two tokens could potentially have been more than ten tokens apart in the original document.

In the language modeling world, these are referred to as *skip-grams*, and are often used to allow a language model to capture additional information about the contents of a text.[6] In our case, we chose to use skip-grams in part to compensate for the large difference in length between the documents and the queries, which can cause problems relating to model sparsity. By allowing this "slop" we are effectively increasing the number of possible bigrams from each query, thereby increasing the number of chances for that query's bigrams to occur in the per-document language models.

Scoring & Ranking Documents Given a normalized query and a normalized set of documents, we compute the log-probability for all pseudo-bigram pairs

[5] We used an absolute discount parameter value of 0.5.

[6] For more details, two good places to start would be the work of Siu & Ostendorf[10] and that of Guthrie, et al.[11].

Run	P@10	MAP	nDCG@10
Traditional (Baseline)	0.2300	0.0953	0.2436
Lang. Model	0.2600	0.0999	0.2344
Baseline w/ MetaMap	0.1620	0.0816	0.1706

Table 1. Official results, including precision at rank 10 (P@10), mean average precision (MAP), and normalized discounted cumulative gain at rank 10 (nDCG@10).

present in the query occurring in each document, just as one would using a standard bigram language model when estimating the log-probability of a sentence. However, in our case, we are not strictly calculating the log-probability of the query vs. each document. Because of our adjusted bigram scheme, each bigram may potentially be counted multiple times depending on how the documents' tokens are arranged; as such, the actual probabilities we calculate are somewhat smaller than they would be in a traditional bigram scheme. However, as we are calculating it in the same way across documents and queries, the probabilities themselves are directly comparable with one another.

As such, in order to rank the documents in order of "relevance," we need simply to rank the documents in descending order of probability (i.e., the documents with the highest probabilities for a given query are ranked highest in the result list). There are any number of different ways to choose a threshold at which to cut off results; we chose a fairly robust method in which we used the histogram of calculated probabilities for a given query to choose a reasonable threshold.

3 Results

Unfortunately, our runs did not perform particularly well this year. Generally speaking, all three of our runs performed below the median in terms of P@10, and none achieved an overall mean average precision score higher than 0.1. However, there were several topics for which we achieved adequate performance, and there were even several topics for which one or the other of our runs appeared to have the best P@10. Our official results can be seen in Table 1.

As is often the case in these kinds of evaluation campaigns, we observed a large amount of variation in system performance across topics (see Figure 1). We also noted a similarly large amount of variation in the number of documents judged relevant for each topic, with one run having zero relevant documents, seven more having fewer than ten, and one topic apparently having 610 relevant documents.[7]

[7] This would be topic `qtest19`, *"is abdominal pain due to helicobacter pylori a symptom of cancer."*

3.1 Traditional System

Our baseline system— essentially plain-vanilla Lucene, without external annotations of any sort— did not perform particularly well. In terms of `bpref`, there were only three topics for which it achieved a score higher than 0.5, and there was a great deal of variability among the remaining topics (see Figure 1). In terms of P@10, its performance was similarly dismal. There were 14 topics for which its P@10 was 0.0; one topic (`qtest29`, *"what is prognosis and the treatment for aortic stenosis"*) came in at 0.8, with a small handful coming in above 0.5. Most of the topics had P@10 scores of between 0.1 and 0.4.

In addition to our baseline system, we also submitted a run from a system that used MetaMap to perform query expansion (see Section 2.1). In previous evaluation campaigns, this approach had often resulted in improved performance. In this task, the picture was somewhat more complex. Often, rather than retrieving *more* potentially-useful results, the query expansion (which made rather aggressive use of Boolean query operators) would result in *fewer* results being retrieved (often many fewer, down to and including zero).

Sometimes, as in the case of `qtest27`, this would result in improved precision (in this case, from a P@10 of 0.0 in the baseline run to 0.6 in the MetaMap run). More often, however, this would result in a decrease in precision, simply due to the query's Boolean criteria preventing more than a handful of articles from being retrieved.

Overall, there were 17 topics for which the MetaMap query expansion *hurt* the system's performance in terms of `bpref`, and 18 for which it had no effect. That leaves 15 for which the query expansion improved `bpref` (see Table 2). In three of these cases (`qtest7`, `qtest27`, and `qtest46`), the MetaMap run's non-zero `bpref` represented an improvement over a baseline `bpref` of 0.0. The remaining cases of improvement fell into two categories. The first consisted of several runs that featured relatively modest improvements (*Metamap*:*Baseline* `bpref` ratios ranging from ≈ 1.06–≈ 1.70). The second consisted of a small number of runs that experienced a much more dramatic improvement, with `bpref` ratios ranging from ≈ 4.40 to ≈ 5.15.

Looking at the final processed form of the queries that experienced the most performance improvement, the thing that stands out is that they are simpler and more parsimonious than they were before being processed by MetaMap. This was not a universal result of being run through our MetaMap processor; quite often, the final queries would be quite large and complex, with multiple Boolean clauses. In these cases of notable improvement, it seems as though MetaMap acted as something of a filter, stripping out un-necessary and noisy terms and replacing them with a small number of salient terms. In our future work, we plan to investigate this phenomenon further.

3.2 Language Model System

Strictly speaking, our language model system (LMS) did not perform particularly well. There were several topics for which we did not retrieve very many results,

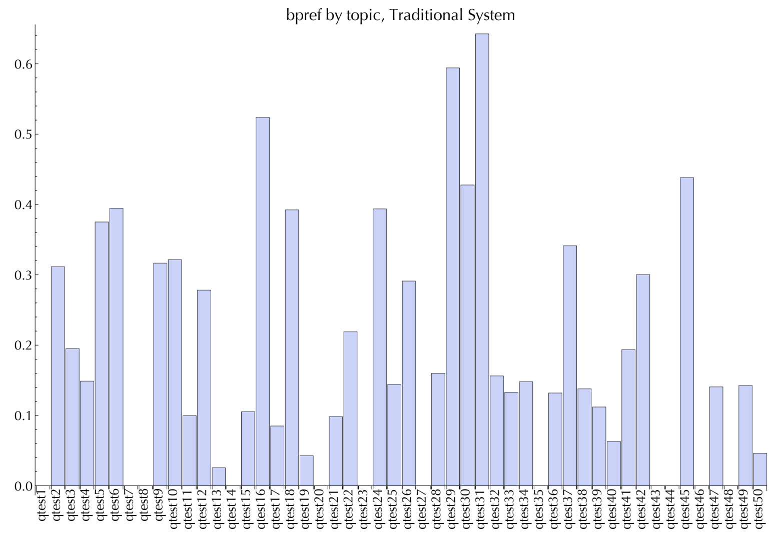

Fig. 1. `bpref` score by topic, "Traditional" (baseline) run. Note the extreme between-topic variability in system performance; note also the large number of topics for which the scored `bpref` was 0.0.

period— much less many relevant results. Overall, our LMS performed even less consistently than did the baseline system (see Figure 2). That said, there were several runs for which the LMS outperformed the baseline system by a wide margin.

For example, for `qtest49`, the LMS:Baseline `bpref` ratio was 3.69, representing a very notable increase in performance. All in all, there were 19 topics for which the LMS outperformed our baseline system in terms of `bpref`, and 7 for which there was no difference in performance. Five of the topics with improved performance saw at least a doubling of `bpref` as compared to the baseline, and one more came very close to double (`bpref` ratio of ≈ 1.97).[8] In terms of P@10, there were six topics for which our LMS run appears to have achieved the best score. In one of these topics (`qtest12`), we appear to have outperformed the median by a fairly significant margin; the other five topics (`qtest20` `qtest31`, `qtest42`, `qtest43`, and `qtest47`) appear to have been more challenging, as the degree to which we outperformed the median was smaller.

One unexpected attribute of the LMS was that diagnosing performance issues was quite difficult. With our baseline system, we can easily examine our generated queries, and can also directly inspect our index. With the LMS, however, things are much more opaque. As such, it is not easy for us to say *why* it did particularly well or badly on any given topic. Future work will explore ways to improve this state of affairs.

[8] On the other hand, there were 25 topics for which the LMS exhibited *decreased* `bpref` performance as compared to the baseline.

Topic	Baseline	MetaMap	Abs. Diff	Rel. Diff
qtest31	0.6426	0.6816	0.039	1.06069
qtest5	0.3751	0.4568	0.0817	1.21781
qtest11	0.0998	0.1315	0.0317	1.31764
qtest30	0.4277	0.5723	0.1446	1.33809
qtest37	0.3412	0.4646	0.1234	1.36166
qtest24	0.3937	0.5728	0.1791	1.45491
qtest32	0.1562	0.2461	0.0899	1.57554
qtest39	0.112	0.1936	0.0816	1.72857
qtest50	0.0463	0.2037	0.1574	4.39957
qtest36	0.1319	0.6019	0.47	4.56331
qtest34	0.1479	0.6805	0.5326	4.60108
qtest17	0.085	0.4375	0.3525	5.14706
qtest7	0	0.1389	0.1389	N/A
qtest27	0	0.4444	0.4444	N/A
qtest46	0	0.0123	0.0123	N/A

Table 2. `bpref` scores for topics that benefited from the MetaMap query expansion described in section 2.1.

4 Conclusions & Future Work

Our results in this year's evaluation campaign were somewhat disappointing. However, we have now established two different solid baseline systems, and have also demonstrated that our existing query expansion system is not completely useless in a more general medically-themed retrieval task. Clearly, however, there is much for us to do in preparation for next year's evaluation campaign. Our future work will consist of work on several fronts.

The first and most important next step will be to improve our indexing strategy. One issue that our baseline system ran into during development was that many documents in the corpus contained extraneous text[9] that led to numerous false positives. We will be working on ways to detect content zones within the page such that we only index the truly relevant text from each document. We also plan on experimenting with various other approaches, such as indexing anchor or heading text separately from the rest of the document body, and boosting their value in the search index.

A second area of work will be in improving our MetaMap-based query parser and expansion system. Right now, it is only using a subset of the vocabularies in the UMLS, which we believe limited its ability to successfully map concepts from the queries into UMLS CUIs. Furthermore, in its present form, the parser is relatively inflexible and makes overly-aggressive use of Boolean operators. This led to over-constrained queries as well as queries containing duplicate terms. We hope to improve this state of affairs.

A third area will be to explore ways to make use of syntactic data derived from the queries themselves to better understand what the queries are actually asking about. This could include constituent or dependency parsing, part-of-

[9] Sometimes in the form of page navigation elements, other times from intentional keyword frequency manipulation on the part of the page's authors— e.g., a hidden `div` filled with a smörgåsbord of medical terms unrelated to the ostensible topic of the page.

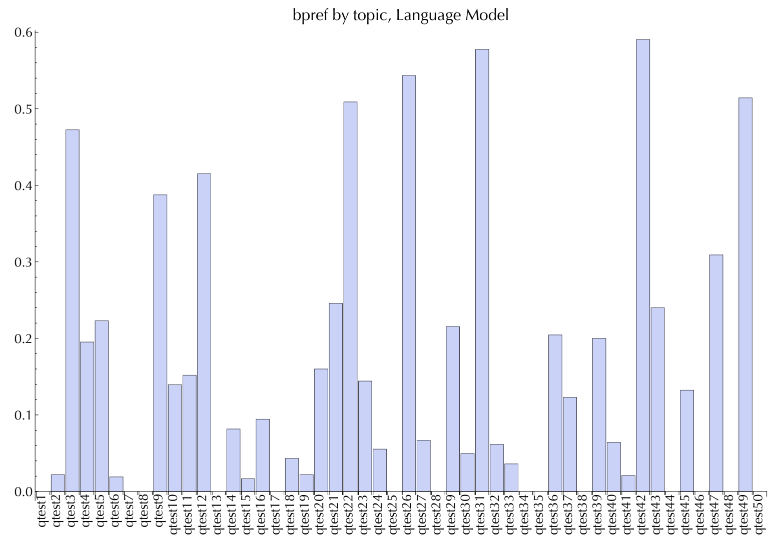

Fig. 2. `bpref` score by topic, "Language Model" run. As compared with the baseline system (see Figure 1), the language model system was less consistent but higher-performing.

speech tagging, and so on. We will also experiment with performing this sort of analysis on the documents as well.

Our language model retrieval system has many possible avenues for improvement. Besides making use of the aforementioned syntactic data, we plan to experiment with different skip-ngram window widths as well as some of the algorithm's other tuning parameters. This sort of optimization work was difficult to do during the development phase of this campaign due to our lack of supervised training data; our hope is that, with access to the qrels from this year's campaign, we will be able to improve our approach.

We are excited to have participated in this year's evaluation campaign, and eagerly await next year's. Although we were disappointed by our systems' performance, we see a lot of potential in our fundamental approach, and look forward to the opportunity to develop our ideas further.

References

1. Voorhees, E., Hersh, W.: Overview of the trec 2012 medical records track. In: The Twenty-First Text REtrieval Conference Proceedings (TREC 2012). (2012)
2. Hersh, W.R., Crabtree, M.K., Hickam, D.H., Sacherek, L., Friedman, C.P., Tidmarsh, P., Mosbaek, C., Kraemer, D.: Factors associated with success in searching MEDLINE and applying evidence to answer clinical questions. J Am Med Inform Assoc **9**(3) (2002) 283–93
3. Suominen, H., Salanterä, S., Velupillai, S., Chapman, W.W., Savova, G., Elhadad, N., Mowery, D., Leveling, J., Goeuriot, L., Kelly, L., Martinez, D., Zuccon, G.:

Overview of the ShARe/CLEF eHealth Evaluation Lab 2013. In: CLEF 2013. Lecture Notes in Computer Science (LNCS), Springer (2013)

4. Bedrick, S., Edinger, T., Cohen, A., Hersh, W.: Identifying patients for clinical studies from electronic health records: TREC 2012 medical records track at OHSU. In: The Twenty-First Text REtrieval Conference Proceedings (TREC 2012). (2012)
5. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA (2008)
6. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: OSDI'04: Sixth Symposium on Operating System Design and Implementation, USENIX Association (2004)
7. Aronson, A.R., Lang, F.M.: An overview of metamap: historical perspective and recent advances. J Am Med Inform Assoc **17**(3) (2010) 229–36
8. Bodenreider, O.: The unified medical language system (umls): integrating biomedical terminology. Nucleic Acids Res **32**(Database issue) (Jan 2004) D267–70
9. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to information retrieval. [5] chapter Language Models for Information Retrieval
10. Siu, M., Ostendorf, M.: Variable n-grams and extensions for conversational speech language modeling. Speech and Audio Processing, IEEE Transactions on **8**(1) (2000) 63–75
11. Guthrie, D., Allison, B., Liu, W., Guthrie, L., Wilks, Y.: A closer look at skip-gram modelling. In: Proceedings of the 5th international Conference on Language Resources and Evaluation (LREC-2006). (2006) 1–4