Trading accuracy for faster entity linking

Kristy Hughes **Joel Nothman** **James R. Curran**
ə-lab, School of Information Technologies
University of Sydney
NSW 2006, Australia
{khug2372@uni.,joel.nothman@,james.r.curran@}sydney.edu.au

Abstract

Named entity linking (NEL) can be applied to documents such as financial reports, web pages and news articles, but state of the art disambiguation techniques are currently too slow for web-scale applications because of a high complexity with respect to the number of candidates. In this paper, we accelerate NEL by taking two successful disambiguation features (popularity and context comparability) and use them to reduce the number of candidates before further disambiguation takes place. Popularity is measured by in-link score, and context similarity is measured by locality sensitive hashing.

We present a novel approach to locality sensitive hashing which embeds the projection matrix into a smaller array and extracts columns of the projection matrix using feature hashing, resulting in a low-memory approximation. We run the linker on a test set in 63% of the baseline time with an accuracy loss of 0.72%.

1 Introduction

Named entity linking (NEL) (Bunescu and Pasca, 2006; Varma et al., 2009; Cucerzan, 2007) is the task of mapping mentions of named entities to their canonical reference in a knowledge base (KB). Recently, this task has been motivated by the Text Analysis Conference (TAC) Knowledge Base Population (KBP) entity linking task. In this task, systems are given queries comprising of a mention string and document ID, and return the referring entity ID from the KB (in this case, Wikipedia) or a NIL ID if the entity is not in the KB.

While large amounts of candidate data provides rich information that can be used for disambiguation, we show that data transfer and query costs comprise over 51% of running time in the unsupervised configuration of Radford (2014). Additionally, whole document approaches for disambiguation compare text using methods such as cosine similarity, but are slow for high-dimensional data such as text (Indyk and Motwani, 1998).

In this paper, we pre-filter candidates using a combination of candidate popularity and context similarity. These features are complimentary to each other, because a high in-link count suggests that the mention string refers to that entity frequently, but a high similarity measure is required for less popular candidates. Since context comparison is expensive to compute, we use LSH (Gionis et al., 1999) to produce a compact binary hash representation of each document, which can be compared using Hamming distance. This fast, informed pre-filtering reduces data communication costs and the number of similarity comparisons.

While LSH has been shown to be a good approximation to cosine similarity on image data (Lu et al., 2008), we show that it this is also the case on text data. Using a hash size of 1024 bits, LSH similarity and cosine similarity have a Spearman correlation of 0.94, with correlation increasing as hash size increases. Using a hash size of 2kB and a LSH similarity threshold of 0.56, and keeping the top 7 candidates by in-link score, we run the linker in 63% of the time with an accuracy loss of 0.72%.

We also present a new, low-memory version of LSH which embeds the projection matrix into a smaller vector and extracts rows of the projection matrix using feature hashing. This method allows for an expandable vocabulary, and only requires storing a single, smaller vector, resulting in fast generation of document hashes.

Our LSH pre-filtering method enables the task to feasibly be applied to linking longer documents (financial reports), big data (the web), and real-time or frequently updated documents (the news) with only a very small drop in accuracy.

2 Background

Named entities (NEs) in natural language are often difficult to resolve; one entity can be referred to by many different mention strings (synonymy) or multiple distinct entities referred to by the same mention string (polysemy). While the task of resolving this ambiguity is automatically and subconsciously performed by most people when they read text, this is a much more difficult task for automated systems to perform.

The NE ambiguity problem has been approached within the field of computational linguistics as three related tasks: Cross-document Coreference Resolution (Bagga and Baldwin, 1998), Wikification (Mihalcea and Csomai, 2007), and named entity linking (NEL) (Bunescu and Pasca, 2006). NEL aims to link in-text mentions of NEs to a knowledge base (KB) using the context of the mention and the vast amount of structured and unstructured information held in the KB.

Approaches to NEL vary (Hachey et al., 2013; Ji and Grishman, 2011), however many systems share some core components. Radford et al. (2012) combines three seminal approaches (Cucerzan, 2007; Varma et al., 2009; Bunescu and Pasca, 2006) to produce the NEL system on which this paper is based. Almost all approaches can be split into 3 stages: mention extraction, candidate generation and candidate disambiguation.

The mention extraction stage involves chaining together mentions in a document that refer to the same entity, and candidate generation stage involves retrieving entities from the KB that have similar names to mentions in the query's chain. The candidate disambiguation stage compares each candidate with the query and ranks them by the aggregate of similarity scores.

Core to almost all approaches is the usefulness of the prior probability of a candidate entity, and the amount of overlap in text between the query document and candidate entity. These two features are complementary to each other.

Prior probability is a measure of the popularity of the entity. This can be calculated from a large corpus of disambiguated entities, most often using Wikipedia's internal links, and is independent from the query document, so it can be precomputed and stored. Popular entities appear in all contexts, and so they will often not require a particular context for readers to know what the mention string refers to.

Less popular entities are distinguished by readers through the context that they appear. Therefore context similarity is an important measure of the validity of a candidate. The most popular way for comparing the similarity of text is by using cosine similarity, and many of the scores in the candidate disambiguation stage use cosine similarity, some only over sentences and others over entire documents.

To compute cosine similarity over a document, text is mapped to a bag of words (BOW) vector containing the count of each word in the document, and the dot product of these vectors represents their similarity. For a BOW, the dimensionality of the vector is number of words in the vocabulary, v. Cosine similarity can also be applied more broadly using any textual feature as a dimension of the vector. Since these vectors are generally sparse (you do not have every word in the vocabulary appear in one document), it is are often fast to compute. However, due to the high number of comparisons needed (every document-candidate pair), it can become an expensive measure to use.

Dimensionality reduction of these BOW vectors before computing cosine similarity further decreases its computational complexity and the cost of data transfer (as lower dimensional forms can be precomputed and stored). Popular methods of dimensionality reduction are singular value decomposition and principle component analysis (Muflikhah and Baharudin, 2009; Lin et al., 2011), however these are expensive to compute and adds skew to the data, so they don't correlate well with cosine similarity.

Dimensionality reduction through random projections removes much of this pre-computational work, while also not introducing significant skew (Lu et al., 2008). BOW vectors are mapped to lower dimensions by pre-computing some randomly generated hyperplanes called the projection matrix, and computing the matrix multiplication of the vector and the projection matrix.

Locality sensitive hashing (LSH) makes the comparison process in the projected space more efficient by binarising the embedded vectors into a hash (Gionis et al., 1999), approximating cosine similarity of the BOWs as the Jaccard similarity of the hashes. While LSH is faster to compute, it still requires a large projection matrix to be stored, and there needs to be a way of dealing with new words that appear, which we discuss in section 5.

Step	Approx Time (%)	Substep	Approx Time (%)
Initialisation and output	0.9		0.9
Mention Extraction	2.5	C&C NER	1.6
		Chaining	0.9
Candidate Generation	31.0	Build query	0
		Expand query	0.9
		Retrieve candidate ID s	5.7
		Retrieve processed candidates	20.6
		Compile in-memory structure	3.8
Candidate Disambiguation	65.6	In-link prior probability	0.1
		Reference probability	13.1
		Alias cosine	10.6
		Category score	6.2
		Context score	29.5
		In-link overlap	3.2
		Sentence context	2.3
		Rank and determine NIL	0.6
Total Time	100.0		100.0

Table 1: A profile of the TAC 11 dataset reveals that both the candidate generation and candidate disambiguation phases are slow

3 NEL Profile

This paper extends the unsupervised NEL system introduced by Radford (2014) with the aim of increasing its speed while maintaining comparable accuracy. In order to do this, it is important to first discover which stages are computationally expensive. The system consists of three main stages: mention extraction, candidate generation and candidate disambiguation.

The mention extraction stage aims to find all mentions of named entities in a document in order to find aliases for the mention string. It begins by preprocessing the document with a part of speech tagger and performing named entity recognition. Similar to the task of word sense disambiguation, we assume that mention strings have one sense per discourse. Thus, they are clustered into chains that refer to the same entity using limited coreference resolution rules such as acronym expansion and substring name matching.

The candidate generation stage produces a list of candidate entities (Wikipedia articles) for each chain in the document. We take the longest mention string in a chain to be the canonical mention and use this to search the database for suitable matches. Candidates are returned from a Solr database (limited to 100 candidates) if the canonical mention matches matches a Wikipedia pages' title, redirect titles or apposition stripped title. This stage aims to ensure that the correct entity is returned as a candidate, while minimising the size of the candidate set for feasible disambiguation. Once a list of candidate names is obtained from Solr, the data associated with each candidate is retrieved from a Hypertable database.

While these first two stages aim to maximise recall, the candidate disambiguation stage aims to distinguish the correct entity from the rest of the candidates by ranking them according to a score. The final score is the sum of the in-link prior probability, reference probability, alias cosine similarity, category score, context score, in-link overlap and sentence context. A more detailed description of each can be found in Radford (2014).

We profiled (Radford, 2014) to discover linking bottlenecks. It is often difficult to judge timing of systems due to the variability caused by differing hardware and loads, so we ran all our experiments over the TAC 11 dataset and queries 10 times to get an average run-time of 1778.39 seconds with a standard error of 9.5 seconds. Our experiments were run on an unloaded machine with two 2.30GHz Intel(R) Xeon(R) E5-2470 CPUs and 62GiB RAM. The full break-down of this profile can be seen in Table 1, which shows that both the candidate generation and candidate disambiguation steps are expensive.

The candidate generation step is expensive due to an external database query which retrieves candidate information such as Wikipedia pages. The run-time of this step is linear to the number of candidates retrieved, as well as linear to the amount of data associated with each candidate.

The candidate disambiguation stage takes 65.6% of run-time. The most expensive score to compute is the context score, which take context information from the candidate, such as the disambiguating term in each candidate's Wikipedia title, anchor text from links within the first paragraph, and links to pages that link back to the candidate. These context terms are searched for within the query document using a trie. While the complexity of this step is related to the number of context terms, the dominating cost is from the number of candidates. Other expensive steps are the reference probability score and alias cosine score.

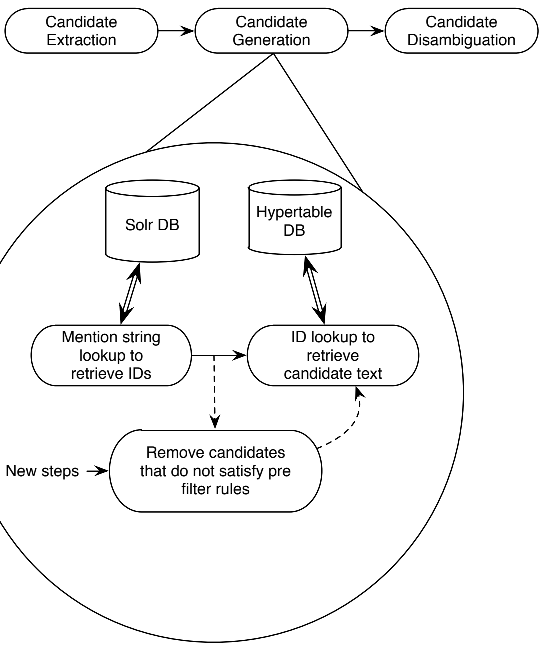

Candidate Extraction
Candidate Generation
Candidate Disambiguation
Solr DB
Hypertable DB
Mention string lookup to retrieve IDs
ID lookup to retrieve candidate text
New steps
Remove candidates that do not satisfy pre filter rules

Figure 1: NEL Pipeline and our changes

4 Pre-filtering candidates

Since the expensive steps that have been identified are costly for each candidate, we consider a pre-filter which reduces the number of candidates before their full text is retrieved from the database, and before disambiguation occurs. This will trade accuracy for speed because the load is reduced for both the candidate generation and candidate disambiguation steps, at the cost of some true candidates inadvertently being eliminated. We hypothesise that correct candidates either are either contextually similar with the query document, or are popular and so do not require any contextual overlap. For example, a query document mentioning Melbourne is likely to have similar words to the Wikipedia page for Melbourne. This contextual similarity is important for capturing the correct entity because Melbourne could also refer to Melbourne, Nova Scotia or Melbourne, Quebec. However, a query document mentioning Australia is not necessarily going to share context with the page for Australia because of its notability.

Popularity is measured using an in-link score, which measures the prior probability that a particular candidate is linked to by the mention string. Since this is document insensitive, in-link scores have been precomputed for all candidates in the KB and are retrieved from the database when the name query occurs. We use the rank of candidates sorted by in-link score because it is more meaningful than their raw score, which may vary greatly between candidates. Since Solr returns candidates sorted by this score in the existing system, using their rank is not an expensive step.

Context between documents is generally measured using cosine similarity. For our purposes, we compute the cosine similarity of the BOW of the document (a vector containing the count of each word in the document). We use these raw counts, rather than TF-IDF because some initial experiments suggested that it did not change the spread of correct candidates and it was expensive to retrieve IDF scores. While we found in-link rank to be a very fast pre-filtering method, cosine similarity has to be computed for each candidate of a given document, making it slow when there is a high number of candidates.

For each query, we retain the top i candidates by in-link count. For any remaining candidates, we calculate the similarity between the query document and each candidate's Wikipedia page text, retaining those with a similarity above ℓ. This fits within the candidate generation stage. We store candidate data needed for these thresholds in the Solr database, so that filtering occurs after the Solr query, but before the Hypertable query (Figure 1). In order for this to be an effective pre-filter, it must be fast with respect to the number of candidates, otherwise it defeats the purpose. Since cosine similarity is slow, we use locality sensitive hashing to approximate cosine similarity.

$$\underbrace{\begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,v} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ r_{b,1} & \cdots & \cdots & r_{b,v} \end{bmatrix}}_{\substack{b\times v \text{ random hyperplanes matrix} \\ H}} \times \underbrace{\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_v \end{bmatrix}}_{\text{BOW}} = \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_b \end{bmatrix}}_{\text{Projected vector}} \rightarrow \begin{matrix} x_i \geq 0 \mapsto 1 \\ x_i < 0 \mapsto 0 \end{matrix} \rightarrow \underbrace{\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 1 \end{bmatrix}}_{\text{Hash}}$$

Figure 2: Creating a document hash representation using LSH

5 Low memory LSH

Locality sensitive hashing (LSH) (Broder, 1997; Indyk and Motwani, 1998) is generally used for grouping similar objects from a large data set by mapping them to buckets based on the bits in their lower-dimensional hash (Ravichandran et al., 2005). Since the number of candidates per document is relatively small, this approach is unnecessary, but we use it to compute an extremely efficient approximate similarity function. This is done by counting the number of bits that are the same between the document and query hash. Since hashes are binary, this can be done taking the XOR of the hashes, and counting the 0's which is very fast using the popcount CPU instruction. While the similarity function is very fast, we also need the hashing technique to be very fast, since documents are unseen and their hashes must be calculated in real-time. To do this, we present a new method of generating hashes which is different from the traditional method.

Traditionally, cosine LSH projects the document vectors, or bags of words (BOWs), of dimension v to a lower dimensional (b) binary hash. A BOW document representation is a real valued vector where each element corresponds to the count of each word in the combined vocabulary of all the documents, excluding stop words. BOWs are projected by computing their dot product with a projection matrix (M) of normally distributed random numbers (r_i). The result of this operation, a low-dimensional vector, is then binarised into a hash by mapping non-negative numbers to 1, and negative numbers to a 0 (Figure 2).

This method of LSH requires the vocabulary to be precomputed, which is not suitable for many NEL applications as unseen documents often contain unseen words that must be dealt with. If unseen words are discarded, the hashes no longer become a true representation of the document. Conversely, adding a row to the projection matrix every time a new word is discovered, which is frequent under Zipf's Law, incurs a runtime cost, and results in unbounded memory consumption. Additionally, query documents are unseen so their hash representation cannot be pre-computed, requiring M to be loaded into memory. This can be expensive with a large vocabulary and high number of bits, as $|M| = b \times v$.

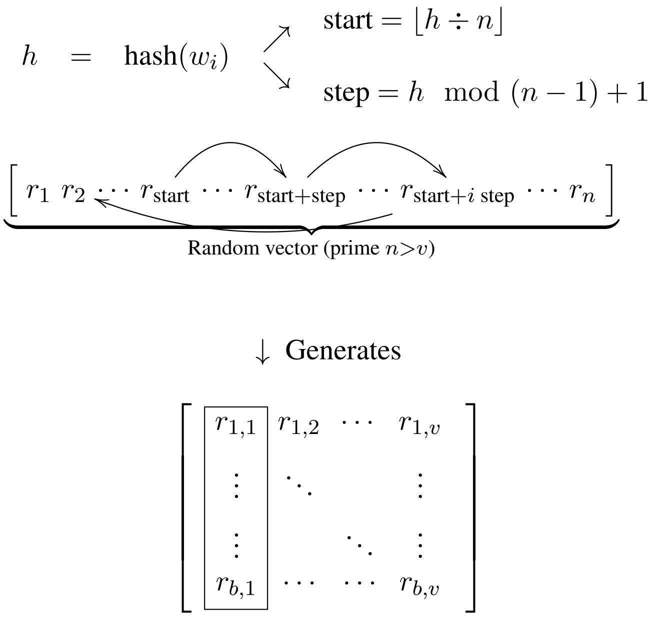

$h = \text{hash}(w_i)$
start $= \lfloor h \div n \rfloor$
step $= h \mod (n-1)+1$
r_1 r_2 $\cdots$ r_{start} $\cdots$ $r_{\text{start+step}}$ $\cdots$ $r_{\text{start}+i\text{ step}}$ $\cdots$ r_n
Random vector (prime $n>v$)
↓ Generates
$r_{1,1}$ $r_{1,2}$ $\cdots$ $r_{1,v}$
$r_{b,1}$ $\cdots$ $\cdots$ $r_{b,v}$

Figure 3: Low-memory LSH generates rows of the hyperplane matrix

We present a low-memory LSH technique which embeds M in a single, fixed-length array, M', and artificially generates rows of M by stepping through M' (Figure 3). To find the start and step for a particular row, the word associated with that row is hashed into a 32-bit integer using a string hash function, xxhash. We divide the integer by the length of M', with the quotient being the starting place in M' and the remainder being the step size. We step through the M' until we have produced a b length row and multiply it by the value corresponding to that word. This is done for each word in the document and the resulting rows are added together and binarised to produce the hash. This effectively mimics the normal matrix multiplication.

The theoretical basis for this method relies on the idea that each word in the document will correspond to a unique start and step, which produces a unique row of the projection matrix. To ensure that no repeating occurs in the generated row, we need the step length to be co-prime to $|M'|$. Thus we choose $|M'|$ to be prime, so that all step lengths are co-prime to $|M'|$. This method allows us to embed an $|M'|(|M'|-1) \times |M'|$ dimension random matrix in M' without substantially affecting LSH, provided that $|M'|$ is prime and larger than the true size of the vocabulary (which is unknown), and $b < |M'|$. This new LSH method only needs to generate and store $|M'|$ random numbers, rather than $b \times v$ random numbers, and thus is more space-efficient than traditional LSH.

6 Experiments and Evaluation

Our first experiment determines the correlation of LSH with cosine similarity to confirm that it correlates well in text data, and to find a suitable hash size. To show the correlation between cosine and LSH, we graphed the cosine similarity and LSH similarity between all query-candidate pairs for the TAC 11 dataset. We calculated both Pearson and Spearman correlation as to not make incorrect distributional assumptions that may affect its validity.

We evaluate our system in terms of both run time and accuracy and measure it as a trade-off, since speed increases usually come at a cost to accuracy. We judge a good trade-off between the time and accuracy as one where the time taken to run the NEL system is significantly shorter without significantly impacting the speed. We use TAC 11 data as a training set, and then test our best configuration on the held-out TAC 12 dataset.

We pre-filter candidates retrieved from the Solr search using the similarity of their hash with the document's hash. This relies on the assumption that candidates with low similarity between documents are likely not to be a correct link. We determine the time-accuracy trade-off when filtering by a similarity threshold. We use in-link score which is retrieved from the Solr database as a baseline pre-filtering method. We also combine both in-link score and LSH similarity to test how they work together. This is under the assumption that in-link score is a measure of entity popularity, and so will retrieve different candidates to LSH similarity, which is an approximation of context similarity.

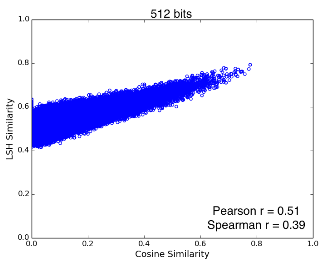
We precomputed the 2kB hashes for all TAC 11 candidates and stored them in the Solr database. Our pre-filtering experiments retrieved the hashes of all candidates during the Solr search (Figure 1), cut-down the hash to the number of dimensions we were experimenting with, and pre-filtered them according to their hash similarity with the query's hash for various thresholds (i.e. hash similarity $>$ threshold). This meant that extra time was added to the candidate generation step by retrieving hashes from Solr and calculating hash similarity, but time was taken away from the candidate generation step also because fewer candidates had to be retrieved from Hypertable. The candidate disambiguation phase is where most of the time gain occurs, as whole document linking has fewer candidates to disambiguate.

The accuracy of the NEL system is the macro-averaged accuracy over the entities, as to align with the TAC task measures. Our experiments are all run over the TAC 11 dataset, and the NIL baseline (linking all queries as NIL) is 51.84. We use the unsupervised configuration of the Radford (2014) system for all of our linking experiments. This configuration took an average of **1778.39** seconds to run (Table 1) and achieved an accuracy **87.16%**. We use this configuration as out baseline for time-accuracy trade-offs that occur when filtering candidates at differing thresholds.

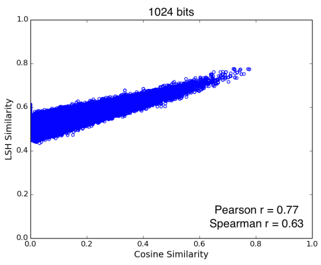
7 Results and Analysis

Our results show that low-memory LSH requires a high number of bits to correlate well with cosine similarity (Figure 4). With 512 bits we can see a linear trend between cosie similarity ane low-memory LSH, however Pearson correlation exceeds 0.9 for hash sizes larger than 2^{13}.

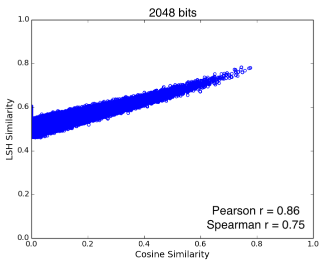
We notice some particularly high hash similarity when the cosine similarity is 0, and after some investigation, we discovered that these similarity scores were for candidates with no text. If a candidate has no text, their default LSH hash is a string of 0's, so their similarity measure with the document is effectively counting the proportion of 0's in the document hash. We expected LSH similarity



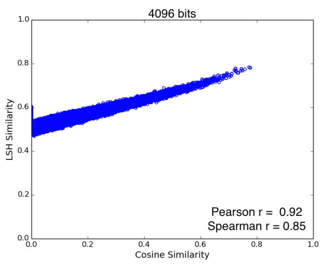
(a) 2^9 bits



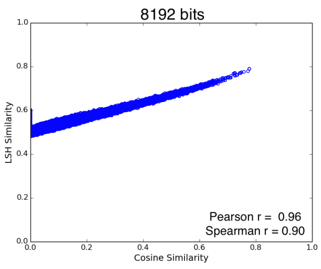
(b) 2^{10} bits



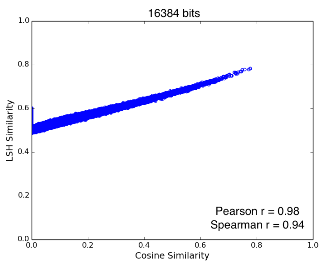
(c) 2^{11} bits



(d) 2^{12} bits



(e) 2^{13} bits



(f) 2^{14} bits

Figure 4: LSH and Cosine correlation increases as hash dimensionality increases.

to be 0.5 for non-correlated documents, since this is the expected value that any two bits are equal. One possible explaination for this relates to the small size of the projection array, M'. With low-memory LSH, each row of M is generated by effectively sampling from a sample (M') rather than the population ($\mathcal{N}$) and so any bias that the sample may have is magnified in the full matrix. This may result in the proportion of 1's having a slight skew away from the theoretical mean of 0.5. Figure 5 shows that the distribution of M' is reasonably centered at 0 for an array of 16 411 random numbers. The results of a t-test to see if the mean was significantly different to 0 was inconclusive, with a p-value of 0.18. However, when we use only 2053 random numbers in M' we see a significant bias. We are not yet sure whether this is the cause of the anomalous points, and whether this theoretical flaw has any practical effect. We have a proposed solution to this problem of the sample mean, which we will discuss later in future work.

In order for the pre-filtering mechanism to be valid, we need the filter method to be very fast and not scale badly with the number of candidates. To use LSH instead of cosine, we need the cost of hashing each document plus the cost of computing the LSH similarity for each document-candidate pair to be faster than the cosine of each document-candidate pair. Our experiments show that LSH similarity performs at 38.1% of the time of cosine

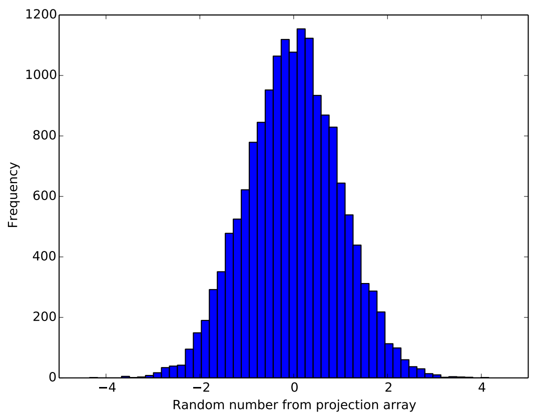


Figure 5: Distribution of M' is not centered at 0. This could be the cause of anomalous LSH similarity scores where cosine similarity is 0

similarity for a hash size of 2^{14} bits. This cost decreases as hash size decreases, however so does the correlation coefficient.

Our time-accuracy trade-offs are shown in Table 2. We notice that there is a general trend with accuracy increasing as the number of top-inlinks increases and as LSH similarity threshold decreases, at the cost of speed. This is also shown visually in Figure 6. This shows us that our best time-accuracy trade-off is when $\ell = 0.56$ and $i = 7$, since only a slight amount of accuracy is lost for a large gain in speed.

Threshold for ℓ	Threshold for i											
	No in-links		1		2		5		7		11	
	Time	Acc	Time	Acc	Time	Acc	Time	Acc	Time	Acc	Time	Acc
No LSH	–	–	42.89	81.33	45.6	84.84	55.53	86.04	58.37	86.53	62.5	86.89
0.53	68.2	84.22	69.76	86.67	71.4	87.2	74.58	87.29	80.27	87.42	79.64	87.47
0.54	56.58	79.87	64.16	86.09	62.61	86.93	66.98	87.07	72.85	87.38	73.76	87.56
0.55	48.78	74.62	55.45	85.38	57.2	86.49	65.66	86.8	69.05	87.16	70.14	87.47
0.56	41.93	68.93	51.98	84.62	54.8	86.36	60.05	86.8	63.91	87.11	67.78	87.47
0.57	37.59	64.8	49.41	83.78	54.16	85.87	58.31	86.62	64.15	86.98	67.58	87.38
0.58	34.97	62.36	49.07	83.47	51.61	85.78	58.1	86.58	63.25	86.98	66.39	87.29
0.59	33.77	59.82	48.19	82.98	51.22	85.51	57.25	86.53	63.06	86.89	66.22	87.2
0.6	32.16	58.13	47.63	82.84	51.75	85.42	56.98	86.49	63.48	86.84	65.91	87.16

Table 2: Time-accuracy trade-off for different in-link ranks and LSH similarity thresholds. Time is measured in percentage of original system (1778.39 seconds) and accuracy is the total accuracy of the system with that configuration

Configuration	Accuracy	Time (s)	Average Time (% of baseline)	Standard Error (%)
Baseline	74.35	2293	100	0.02
$i = 7$ $\ell = 0.56$	73.63	1455	63	0.01
$i = 2$ $\ell = 0.56$	71.52	1179	53	–

Table 3: Results for three chosen thresholds shows that this method is robust across unseen datasets.

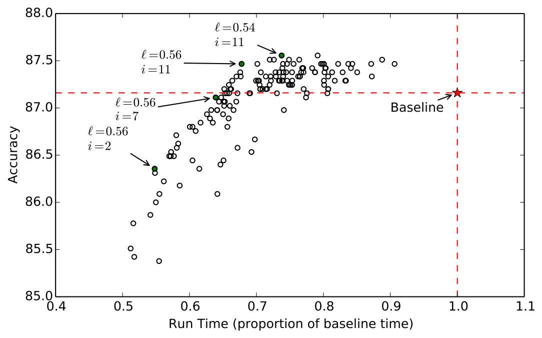


Figure 6: Candidate filtering time for different configurations. Top in-links is fast, while the combined LSH configurations take more time

We choose $\ell = 0.56$ and $i = 7$ to test on an unseen dataset, with results shown in Table 3. We see that results are robust to an unseen dataset, with accuracy decreasing by 0.42 while running at 63.4% of the baseline time.

8 Conclusion

In this paper, we used pre-filtering of candidates to achieve faster time in linking without substantial loss of accuracy. Using a LSH similarity threshold of 0.54 and keeping the top 3 in-links, we decreased the speed by 20% with no loss of accuracy.

We also presented a new method for calculating LSH which runs much faster than regular LSH, requires significantly less storage space than regular LSH and also allows for an expanding vocabulary. We show that this method correlates well with cosine similarity, with a hash size of 1024 bits having a Spearman correlation score of 0.94.

The system we presented uses relatively simple heuristics to decrease the number of candidates that need to be processed in the disambiguation phase. This enables supervised models with large feature sets to be feasibly trained. Our low-memory LSH method can be applied elsewhere in NEL, such as in the disambiguation phase. Features that were previously too expensive in the original vector space can be hashed and their similarity approximated. This is particular useful for features that have a high complexity with regard to candidate size.

Acknowledgments

This work was supported by ARC Discovery grant DP1097291. The authors thank the anonymous re-

viewers and the ə-lab researchers for their helpful feedback.

References

Amit Bagga and Breck Baldwin. 1998. Entity-based cross-document coreferencing using the vector space model. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1*, ACL '98, pages 79–85.

Andrei Broder. 1997. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, pages 21–29.

Razvan Bunescu and Marius Pasca. 2006. Using encyclopedic knowledge for named entity disambiguation. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-06), Trento, Italy*, pages 9–16, April.

Silviu Cucerzan. 2007. Large-scale named entity disambiguation based on Wikipedia data. In *Proceedings of EMNLP-CoNLL 2007*, pages 708–716.

Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529.

Ben Hachey, Will Radford, Joel Nothman, Matthew Honnibal, and James R. Curran. 2013. Evaluating entity linking with Wikipedia. *Artificial Intelligence*, 194:130–150, January.

Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613.

Heng Ji and Ralph Grishman. 2011. Knowledge base population: Successful approaches and challenges. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 1148–1158, Stroudsburg, PA, USA. Association for Computational Linguistics.

Lin Lin, Chao Chen, Mei-Ling Shyu, and Shu-Ching Chen. 2011. Weighted subspace filtering and ranking algorithms for video concept retrieval. *IEEE Multimedia*, 18(3):32–43.

Yu-En Lu, Pietro Lió, and Steven Hand. 2008. On low dimensional random projections and similarity search. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 749–758.

Rada Mihalcea and Andras Csomai. 2007. Wikify!: Linking documents to encyclopedic knowledge. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, CIKM '07, pages 233–242.

Lailil Muflikhah and Baharum Baharudin. 2009. Document clustering using concept space and cosine similarity measurement. In *Proceedings of the 2009 International Conference on Computer Technology and Development - Volume 01*, ICCTD '09, pages 58–62.

Will Radford, Will Cannings, Andrew Naoum, Joel Nothman, Glen Pink, Daniel Tse, and James R. Curran. 2012. (Almost) Total recall. In *Proc. Text Analysis Conference (TAC2012)*.

Will Radford. 2014. *Linking Named Entities to Wikipedia*. Ph.D. thesis, University of Sydney.

Deepak Ravichandran, Patrick Pantel, and Eduard Hovy. 2005. Randomized algorithms and nlp: Using locality sensitive hash function for high speed noun clustering. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 622–629.

Vasudeva Varma, Praveen Bysani, and Kranthi Reddy. 2009. IIT Hyderabad at TAC 2009. In *Proc. Text Analysis Conference (TAC2009)*.