Empiric Introduction to Light Stochastic Binarization

Daniel Devatman Hromada[1,2]

[1] Slovak University of Technology, Faculty of Electrical Engineering and Information Technology, Department of Robotics and Cybernetics, Ilkovičova 3, 812 19 Bratislava, Slovakia
[2] Université Paris 8, Laboratoire Cognition Humaine et Artificielle, 2, rue de la Liberté 93526, St Denis Cedex 02, France

Abstract. We introduce a novel method for transformation of texts into short binary vectors which can be subsequently compared by means of Hamming distance measurement. Similary to other semantic hashing approaches, the objective is to perform radical dimensionality reduction by putting texts with similar meaning into same or similar buckets while putting the texts with dissimilar meaning into different and distant buckets. First, the method transforms the texts into complete TF-IDF, than implements Reflective Random Indexing in order to fold both term and document spaces into low-dimensional space. Subsequently, every dimension of the resulting low-dimensional space is simply thresholded along its 50th percentile so that every individual bit of resulting hash shall cut the whole input dataset into two equally cardinal subsets. Without implementing any parameter-tuning training phase whatsoever, the method attains, especially in the high-precision/low-recall region of 20newsgroups text classification task, results which are comparable to those obtained by much more complex deep learning techniques.

Keywords: Reflective Random Indexing, unsupervised Locality Sensitive Hashing, Dimensionality Reduction, Hamming Distance, Nearest-Neighbor Search

1 Introduction

In applied Computer Science one often needs to select from the database an object which most resembles the "query" object already at one's disposition. In order to do so, all members of the database are often transformed into ordered sequences of numeric values (i.e. vectors). Such vectors can be interpreted as points in the high-dimensional metric space allowing to calculate their distance to other points in the space. In such case, the resulting "most similar" entity is simply the entity whose vector has smaller distance to the vector representing the "query" entity than any other entity stored in the database, i.e is query's "nearest neighbor".

In Natural Language Processing (NLP), the nearest-neighbor search (NNS) is a widely-used approach applied for solving diverse problems. Seemingly trivial, NNS is nonetheless not an easy problem to tackle with, especially in the case of Big Data scenarios where database contains huge amount of highly-dimensional datapoints. In real-time scenarios where naive linear comparison of *d*-dimensional query vector with all *N* vectors stored in the database is simply not feasible due to its *O(Nd)* computational complexity. Thus, one is almost always obliged to take recourse in approximation or heuristic-based solutions.

P. Sojka et al. (Eds.): TSD 2014, LNAI 8655, pp. 37–45, 2014.

One of the most common methods of reducing the complexity of the NN-search is by reducing the dimensionality of the database-representing vector space. Classical approach to do so is Latent Semantic Analysis [10] (LSA). Other family of more and more common approaches exploits so-called binary vectors as the ultimate means of entity's formalisation. Given the fact that contemporary computers are machines essentially -i.e. on the physical hardware level- always working with binary distinctions, the calculation of the distance between two binary vectors (i.e. Hamming distance – the number of times a bit in $vector_1$ has to be flipped in order to obtain the form of $vector_2$) can be indeed a very fast operation to realize, especially when implemented on the hardware level as a part of processor's instruction set.

Combination of dimensionality reduction and binarisation are basis for family of methods descending from the approach called Locally Sensitive Hashing (LSH) [11]. While concrete implementations often substantially differ – c.f. [13] for the state-of-the-art overview – the objective is always the same: to hash each object of the dataset into a concise binary vector in such a way that the objects which are similar shall end up in the same or similar bucket (i.e. shall be represented by same or similar binary vector) while the objects which are disparate shall end up in disparate buckets[3].

In order to attain stunningly good results, many of these methods have to be first trained. Such a tradeoff of high performance / complexity of training phase is the case, for example, in the "semantic hashing" (SH) approach of [1]. In SH one has to first learn the weights between different restricted Boltzmann machines in order to obtain a multi-layered "deep generative model" able to perform the hashing. But the SH has also certain non-negligeable disadvantages: the 1) training-related costs 2) need to work with restricted amount of features which shall enter the first layer of the network (e.g. 2,000 TF-IDF values in [1]) 3) possibility of over-fitting of the model etc.

In this article, we shall present approach, which could one take *vis-a-vis* the problem of "text hashing". Instead of founding our approach on a powerful supervised "deep learning" algorithm able to extract sophisticated combinations of regularities among restricted number of initial features, we shall exploit an algorithm so simple that it can easily integrate huge number of features in a very fast & frugal way. In fact, the algorithm presented here is completely unsupervised and does not need any training or feature-preselection at all in order perform the hashing process.

1.1 Reflective Random Indexing

Theoretically, our approach stems from the lemma of Johnson-Lindenstrauss stating that a small set of points in a high-dimensional space can be embedded into a space of much lower dimension in such a way that distances between the points are nearly preserved [9]. Practically, the JL-lemma was already implemented as so-called Random Projection or Random Indexing algorithms. Random Projection was already quite successfully proposed in relation to the hashing problem [12]. Its much simpler Random Indexing (RI) counterpart, however, was not.

[3] Note that the aim of hashing process as presented in this paper differs substantially from the aim of hashing algorithms like MD5 or SHA2 whose objective is to always hash objects into different buckets.

Since a decade from its initial proposal in [4], RI has already proven its usefulness in regards to NLP problems as diverse as synonym-finding [4], text categorization [5], unsupervised bilingual lexicon extraction [6], implicit knowdledge discovery [2], automatic keyword attribution to scientific articles [3] or measurement of string distance metrics [8].

The basic RI algorithm is quite straightforward to both understand and implement: Given the set of N objects (e.g. documents) which can be described in terms of F features (e.g. occurence of the string in the document), to which one initially associates a randomly generated D-dimensional vector, one can obtain D-dimensional vectorial representation of any object X by summing up the vectors associated to all features F_1, F_2 observable within X. The original random feature vectors are generated in a way that out of D elements of vector, only S among them are set to either -1 or 1 value. Other values contain zero. Since the "seed" parameter S is much smaller than the total number of elements in the vector (D), i.e. S <<D, initial feature vectors are very sparse, containing mostly zeroes, with occasional value of -1 or 1.

At the end of the process, one obtains vectorial characterisations of all documents which one can compare by means of cosine measure. Leaving aside some advantageous properties described elsewhere [8], RI as described here is nothing else than a randomly distorted variation on a "bag-of-words" theme. Consistently to other bag-of-word approaches, one can also weight the initial randomly generated feature vectors with feature's TFIDF [7] value.

But one is not obliged to stop the whole process after the calculation of initial document vectors. One can indeed "reflect" the whole process and proceed this time from object vectors toward feature vectors, forget the initial randomly generated feature vectors of 0^{th} generation and obtain the feature vectors for feature F_X as a sum of vectors O_1,O_2 representing the objects within which one can observe the occurence of feature F_X. Subsequently, the object vectors can be once again calculated as a sum of feature vectors; feature vectors as a sum of object vectors etc. Such a multi-iterative approach whereby every iteration can be potentially followed by vector normalization is called Reflective Random Indexing (RRI).

While RRI keeps the advantageous properties of non-iterative RI like incrementality (i.e. it is very easy to enrich the model with new features or objects) and homogenity (both objects and features are points of the same space), it goes well-beyond simple bag-of-words properties of non-iterative RI. This is so because of certain symmetry in the algorithm where, in every iteration, not only features take part in the vectorial definition of objects but also objects help to construct the vectorial representations of features. Thus, RRI multiplies substantially the amount of mutually interacting forces within the generated metric space and allows for such usages as discovery of "implicit semantic inferences" within huge corpora [2].

Reflective Random Indexing

```
algorithm RRI ()
   #initial iteration is equivalent to plain Random Indexing
   foreach Feature
    Feature_Vectors[Feature] = generate_Random_Vectors(Dimension, Seed)
      Feature_Vectors[Feature] *=  TFIDF_Weights[Feature] #optional
   foreach Object
      foreach Feature in Object2Feature[Object]
         Object_Vector[Object] += Feature_Vectors[Feature]
   normalize Feature_Vectors,Object_Vectors #optional
   #reflective iterations
   repeat
      foreach Feature
         foreach Object in Feature2Object[Feature]
            Feature_Vector[Feature] += Object_Vectors[Object]
      foreach Object
         foreach Feature in Object
            Object_Vector[Object] += Feature_Vectors[Feature]
      Iteration = Iteration + 1
      normalize Feature_Vectors,Object_Vectors  #optional
   until Iteration == MaxIterations
   return Feature_Vectors, Object_Vectors
```

2 Light Stochastic Binarization

Our hashing algorithm is a simple extension of Reflective Random Indexing. While the output of RRI are D-dimensional real-valued vectors, the output vectors of LSB are not real-valued but binary vectors of length D.

Transformation of RRI-generated real-valued vectors into binary vectors is a fairly straightforward process: after all object vectors are calculated by RRI, we simply determine the median value (i.e. 50th percentile[4]) for every dimension (i.e. column) D of the resulting Nd matrix. In such a way we obtain a threshold value for every dimension and we assign into d^{th} element of final binary representation of object *n* the 0 value if its real-valued coordinate along d^{th} dimension is smaller than the determined threshold and 1 if it is above the threshold. Rare tie situations are broken randomly. Result is a set of binary hashes cut in two equally cardinal subsets by every dimension-denoting bit. This binarization is the very last step of the indexing phase.

$$h_d(n) = \begin{cases} 0 & \text{if } n < \text{median}(D_d) \\ 1 & \text{if } n > \text{median}(D_d) \\ \text{rand} & \text{if } n == \text{median}(D_d) \end{cases}$$

[4] Determination of dimension's median value is the only nontrivial component of the process. Note that in case of particularly large and complex samples, law of large numbers shall push 50th percentile's value limitely close to 0.

Subsequently, during the query phase, can simply transform the query object is transformed into its binary vector by: 1)summing up the real-valued representations of the features observable within the query object 2) thresholding the resulting real-vectored by pre-determined medians. Resulting binary vector is subsequently considered to be the center of the Hamming-ball of radius R. Every binary vector contained within such Hamming-ball shall yield an index pointing to the bucket stored in the memory where we could look in order to find query's nearest-neighbor. In case $2^R<B$, i.e. in case when generated Hamming-ball can potentially contain more possibilities than is the total amount B of binary buckets generated during the indexing phase, we can calculate, in a linear fashion, query's Hamming distance H in regards to every bucket-denoting binary vector and subsequently select only those buckets for which H(query_hash,bucket_hash)<R. Radius R is the query-phase thresholding parameter by means of which one can trade precision with recall and vice versa.

3 Experiment

The aim of our preliminary experiment was to assess whether the LSB approach can be useful at all and, if yes, compare the information retrieval faculties of LSB with those of Semantic Hashing. Thus, our results shall be presented in terms of Precision-Recall curves as defined by [1]:

$$Recall = \frac{\text{Number of retrieved relevant documents}}{\text{Total number of all relevant documents}}$$

$$Precision = \frac{\text{Number of retrieved relevant documents}}{\text{Total number of retrieved documents}}$$

Analogicaly to the article with which we compare our data, the retrieved document is considered to be relevant to the query document when they have the same class label.

3.1 Corpus and Pre-processing

In this preliminary work we have confronted the LSB algorithm only with data contained in 20 newsgroups corpus [14]. The corpus contains 18,845 postings taken from the Usenet newsgroup collection divided into training set containg 11,314 postings, rest being the testing set. Both training and testing subsets are divided into 20 different newsgroups which correspond each to a distinct topic. Because our approach aims to introduce an unsupervised hashing scenario, we have left aside the training set and focused all evaluation solely on 7,531 postings contained in the testing set.

Words were extracted from postings by considering every non-word character as a word boundary – 93,591 words were thus extracted, among which 41,782 has occured in more than one posting. These words were considered as features by subsequent RRI. Data were not processed in any other way – no stop words were filtered away and all words which occured in more than one posting were taken into account.

3.2 Empiric Results

In a comlete analogy to simulations performed in [1], we have used every document from the test set as a query document which was compared to all other 7,530 documents. Precision and Recall values were calculated for every query and averaged over all 7,531 queries. Figure 1 compares the Precision – Recall curves of reflective and unreflective variants of LSB with both "Fine-tuned 128-bit Semantic Hashing" and 128-dimensional binarized Latent Semantic Analysis. Non-LSB results are reproduced from Figure 6 of study [1].

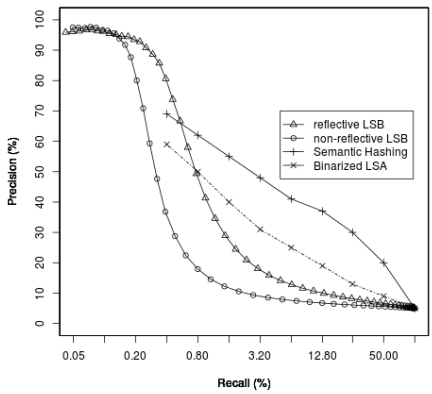

Fig. 1. Comparison of reflective LSB(I=2) and unreflective LSB(I=0) LSB with Semantic Hashing and binarized Latent Semantic Analysis.

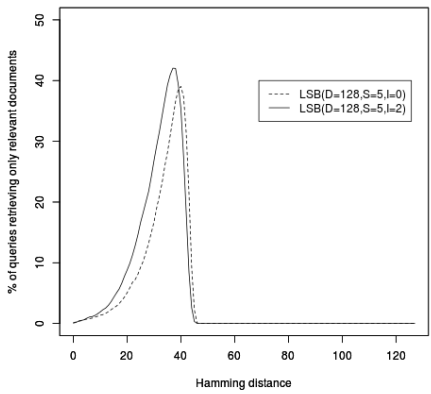

Fig. 2. More than 40% of queries are accompanied, within the Hamming ball of radius 38, only by neighbors belonging to the same newsgroup category.

Figure 2 illustrates in closer detail the behaviour of both reflective and non-reflective variants of LSB in relation to the variation of the retrieval parameter (i.e. Hamming ball's radius R) which is plotted on the X axis. Y axis represents the number of queries which do not have – in their surrounding Hamming ball with radius R – any posting not having the same newsgroup label (i.e. they do not retrieve any false positive), but in the same time, they do retrieve at least one true positive (i.e. at least one object belonging to same newsgroup as query has the binary hash which is located within query's Hamming ball of radius R).

Notwithstanding the size of the theoretically possible hashing search space (2^{128}) which by large exceeds the number of 7,531 initial objects, LSB succeeded to create some collisions, hashing all articles in the 20newsgroup corpus into 7,526 binary buckets. While majority of such "collisions" are due to fairly trivial reposting of the same message, couple of somewhat more divergent messages from comp.graphics newsgroup was also hashed into the same 16-byte bucket. Displayed in the listing below is the difference of content between these two files, as produced by UNIX command diff.

```
< New since version of 2 May 1993:
<   * Added info on ImageViewer for NeXT.
---
> New since version of 18 April 1993:
>   * New version of XV supports 24-bit viewing for X Windows.
>   * New versions of DVPEG & Image Alchemy for DOS.
>   * New versions of Image Archiver & PMView for OS/2.
>   * New listing: MGIF for monochrome-display Ataris.
461,463c464,466
<     PMView 0.85: JPEG/GIF/BMP/Targa/PCX viewer.  GIF viewing very fast,
<     JPEG viewing roughly the same speed as the above two programs.  Has
<     image manipulation & slideshow functions.  Shareware, $20.
---
>    PMView 0.85: JPEG/GIF/BMP viewer.  GIF viewing very fast, JPEG viewing
>     fast if you have huge amounts of RAM, otherwise about the same speed
>    as the above programs.  Strong 24-bit display support.  Shareware, $20.
632,641d634
< NeXT:
<
< ImageViewer is a PD utility that displays images and can do some format
< conversions.  The current version reads JPEG but does not write it.
< ImageViewer is available from the standard NeXT archives at
< sonata.cc.purdue.edu and cs.orst.edu, somewhere in /pub/next (both are
< currently being re-organized, so it's hard to point to specific
< sub-directories).  Note that there is an older version floating around that
< does not support JPEG.
```

In spite of difference of their contents, files comp.graphics/39638 and comp.graphics/39078 of 20-newsgroup corpus, LSB assigned to them the same "10010011000010111001100 01011001000111100001101110100100101001101011101000000010010100110110 01001001010100110100001011111011011" hash

4 Discussion

Looking at the peak shown in Figure 2, one is tempted to state that when confronted with data from the testing set of 20newsgroups corpus, the reflective 128-dimensional LSB is able to retrieve, in 42% of cases (i.e. 3,166 out of 7,531), at least one relevant "neighbor" with maximal precision. It is indeed at the Hamming distance 38, where the method combining Reflective Random Indexing executed with parameters D=128, S=5, I=2 and followed by simple binary thresholding of every dimension, attains at overall recall rate 0.39%[5] to much higher precision (80.6%) than any method presented in the study of [1].

[5] We precise that when we mention 42% recall with 100% precision, we speak about NNS scenario where it is sufficient for a query to retrieve one relevant document. This scenario is documented on Fig. 2. On the other hand, when we mention attained recall rate 0.39%, we speak about much more difficult "classification" scenario where query, in order to attain maximal recall, must retrieve all >370 postings which belong to the same class. This scenario is documented on Figure 1.

On the other hand, LSB performs much worse than compared methods in situations where one wants to attain higher recall. This is most probably due to almost complete lack of "training" – since with exception of 1) TFIDF weighting of initial randomly-generated feature vectors 2) the "reflection" procedure which aids us to characterize objects in terms of features and features in terms of objects 3) determination of binary thresholds (i.e. medians) – there is no kind of "learning" procedure involved.

But it might be the case that lack of any complex "deep learning" procedure shall prove itself to be a certain advantage. More concretely, the one who uses LSB is not obliged to drastically reduce the number of features by means of which all objects are characterized. Thus, in the case of the introductory experiment presented in this paper, we have represented every text as a linear combination of vectors associated to 41,782 features. We believe that it is indeed this ability to "exploit the minute details" (compare to 2,000 words with highest TFIDF score used by [1]) which allows the method hereby introuced to attain higher precisions in scenarios where just one relevant document is to be retrieved. It would be, however, unfair to state that can LSB "pefroms better" than Semantic Hashing, because the goal purpose of SH was not to target the NN-search problem but to yield robust results in more exhaustive classification scenarios. Thus, comparison of LSB with other methods is needed.

It might also be the case that more advanced tuning of RRI's parameters could improve the performance. Another possible direction of research is to study the impact of strategies by means of which the initial random vectors are weighted. Due to the introductory nature of this paper, not much was unveiled about neither of two problems. Looking at the Figure 1, one can, however, assert that: 1) LSB seems to attain better results when its RI component involves more than one iteration, i.e. when it is "reflective".

In sum, we believe that the method hereby introduced is worth to be studied somewhat further. Not only because its dimensionality-reduction component – the RRI – is less costly and more opened to incremental addition of new data than, for example, LSA [10]. Not only because it is similar to LSH [11] in its ability to transform texts into hashes as big as concise as 16 ASCII characters and yet preserve the relations of similarity and difference held by original texts. But also because the algorithm is easy to comprehend, simple to implement and queries can be very fast to execute. That's why we label the method of binarization hereby presented as not only stochastic, but also light.

References

1. R. Salakhutdinov et G. Hinton, "Semantic hashing", International Journal of Approximate Reasoning, vol. 50, no. 7, p. 969–978, 2009.
2. T. Cohen, R. Schvaneveldt, et D. Widdows, "Reflective Random Indexing and indirect inference: A scalable method for discovery of implicit connections", Journal of Biomedical Informatics, vol. 43, no. 2, p. 240–256, 2010.
3. A. El Ghali, D. Hromada, et K. El Ghali, "Enrichir et raisonner sur des espaces sémantiques pour l'attribution de mots-clés", JEP-TALN-RECITAL 2012, p. 77, 2012.
4. M. Sahlgren et J. Karlgren, "Vector-based semantic analysis using random indexing for cross-lingual query expansion", In: Evaluation of Cross-Language Information Retrieval Systems, 2002, p. 169–176.

5. M. Sahlgren et R. Cöster, “Using bag-of-concepts to improve the performance of support vector machines in text categorization”, In: Proceedings of the 20th international conference on Computational Linguistics, 2004, p. 487.
6. M. Sahlgren, “An introduction to random indexing”, In: Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering, TKE, 2005, vol. 5.
7. C. D. Manning, P. Raghavan, et H. Schütze, Introduction to information retrieval, vol. 1. Cambridge University Press Cambridge, 2008.
8. D. D. Hromada, “Random Projection and Geometrization of String Distance Metrics”, In: Proceedings of the Student Research Workshop associated with RANLP, 2013, p. 79–85.
9. W. B. Johnson et J. Lindenstrauss, “Extensions of Lipschitz mappings into a Hilbert space”, Contemporary mathematics, vol. 26, no. 189–206, p. 1, 1984.
10. T. K. Landauer et S. T. Dumais, “A solution to Plato’s problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge.”, Psychological review, vol. 104, no. 2, p. 211–240, 1997.
11. A. Gionis, P. Indyk, et R. Motwani, “Similarity search in high dimensions via hashing”, In: VLDB, 1999, vol. 99, p. 518–529.
12. M. S. Charikar, “Similarity estimation techniques from rounding algorithms”, In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, 2002, p. 380–388.
13. A. Andoni et P. Indyk, “Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions”, In: Foundations of Computer Science, 2006. FOCS ’06. 47th Annual IEEE Symposium on, 2006, p. 459–468.
14. 20 newsgroups, `http://qwone.com/~jason/20Newsgroups/`