Encoding transliteration variation through dimensionality reduction: FIRE Shared Task on Transliterated Search

Parth Gupta[1], Paolo Rosso[1], and Rafael E. Banchs[2]

[1]Natural Language Engineering Lab
Department of Information Systems and Computation
Universitat Politècnica de València, Spain
http://www.dsic.upv.es/grupos/nle
{pgupta,prosso}@dsic.upv.es
[2]Institute for Infocomm Research, Singapore
rembanchs@i2r.a-star.edu.sg

Abstract. There exist a large amount of user generated Web content in Roman script for the languages which are written in indigenous scripts for various reasons. In the light of this phenomenon, the search engines face a non-trivial problem of matching queries and documents in transliterated space where transliterated content contain extensive spelling variation. This paper describes our proposed method to handle such variation through non-linear dimensionality reduction techniques. The approach achieves MRR as high as 0.84 on the main task of *ad hoc* retrieval.

1 Introduction

A large number of languages, including Arabic, Russian, and most of the South and South East Asian languages, are written using indigenous scripts. However, often the websites and the user generated content (such as tweets and blogs) in these languages are written using Roman script due to various socio-cultural and technological reasons. This process of phonetically representing the words of a language in a non-native script is called transliteration. Transliteration, especially into Roman script, is used abundantly on the Web not only for documents, but also for user queries that intend to search for these documents. More often there exist multiple Roman script transliterations for the native terms; for example, the word *Pahala* ("First" in Hindi and many other Indian languages) can be written in Roman script as *pahalaa, pehla, pahila, pehlaa, pehala, pehalaa, pahela, pahlaa, pahelo* and so on. This phenomenon presents a non-trivial term matching problem for search engines to match the vernacular script query with the documents in multiple scripts taking into account extensive spelling variation. This problem can be split into two sub-problems:

1. Handle spelling variation in same script
2. Forward/Backward-transliteration[1]

[1] Backward transliteration means English→Hindi transliteration for the original word in Hindi *e.g. Pyar* → प्यार, while Forward transliteration is Hindi→English for the original word in Hindi.

In this study, we address the issue of *ad hoc* retrieval in transliteration space where documents contain songs lyrics in Roman and Devanagari scripts while the queries are in Roman script.

2 Task

The ultimate aim of the task is to retrieve relevant documents in vernacular scripts and transliterated Roman script with respect to the Roman script queries. The task is divided into two sub-tasks,

2.1 Subtask-1: Query Word Labeling

In this subtask, the participants are given queries in Roman script. Participants have to identify if each of these words are either English words or transliterated in Roman script from one of the Indian languages. If the word is transliterated, a correct transliteration in native script should be provided if the term is not a named entity of type *Person* or *Location*. This subtask is considered as one of the first steps before one can tackle the bigger problem.

2.2 Subtask-2: Multi-script Ad hoc retrieval for Hindi Song Lyrics

In this subtask, the participants are given a collection of documents that contain Bollywood songs lyrics in Roman and Devanagari script and queries in Roman script. The task is to retrieve relevant songs. This subtask is considered as a perfect and practical example of transliterated search. For more details on the task description, please refer to the track overview paper [1].

3 Our Approach

As discussed before, there exist valid/invalid but often popular transliteration equivalents of the term in Roman script or in the original script of the language. The algorithm should normalise such spelling variation across the scripts. The simple edit distance based approaches to find near-transliteration leads to very different terms, *e.g. Pahala vs. Pagala*, *Apne vs. Sapne* and many more. In the past there have been some attempts to normalise the spelling variation to aid retrieval algorithms based on sophisticated versions of edit distance like *Editex* which uses Soundex and phoneme based distance [2]. The work done towards transliteration mining is also relevant for this task especially to when used as backward transliteration mining [3–6]. Although algorithms motivated with such hypotheses are quite effective they are not naturally suitable for this task. The limitations of methods like *Editex* are: such methods are valid for one script only at a time and involve high linguistic knowledge of that language. The methods for transliteration mining are motivated to find mappings across the scripts.

In this task, the algorithm is exposed to two major issues:

1. The terms of the native language are written in multiple ways in the Roman script, *e.g.* लम्हा (*Lamha*) is frequently written as *Lamha, Lamhaa, Lamaha* and so on.
2. There also exist also phonetic variation of the terms in the indigenous script: popular but not necessarily correct, *e.g.* मोहब्बत (*Mohabbat*) is also frequently used as मुहब्बत (*Muhabbat*) and महोब्बत (*Mahobbat*).

Our algorithm tries to handle above issues by deriving associations among the characters within and across the scripts jointly. For example, there exist two kind of character level associations,

1. Intra script, *e.g.* s→sh, f→ph, j→z, मु (mu) → मू (moo)
2. Inter script *e.g.* k→क, kh→ ख

Ideally the algorithm should automatically derive such mappings and, given a term in either script, be able to find its intra/inter script equivalents. The core of our algorithm lies in jointly learning such intra/inter script mappings.

3.1 Learning Algorithm

Our learning algorithm is based on non-linear dimensionality reduction techniques. We train a deep autoencoder to learn the character-gram level mappings among inter/intra script words jointly. The autoencoder is a network which tries to learn an approximation of the identity function so as the output is similar to input. The autoencoder approximates the identity function in two steps: *i)* reduction, and *ii)* reconstruction. The reduction step takes the input $x \in \mathbb{R}^n$ and maps it to $y \in \mathbb{R}^m$ where $m < n$ which can be seen as a function $y = g(x)$ with $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$. On the other hand, the reconstruction step takes the output of the reduction step y and maps it to $\hat{x} \in \mathbb{R}^n$ in such a way $\hat{x} \approx x$ which is considered as a $\hat{x} = f(y)$ with function $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$. The full autoencoder can be seen as $f(g(x)) \approx x$. If the bottleneck dimension (m) is sufficiently small, the autoencoder is able to learn very powerful and meaningful abstract features.

The architecture of the autoencoder is shown Fig. 1 (a). Here each level is a restricted boltzmann machine (RBM) which contain a visible layer and hidden layer. Such RBMs are stacked on top of each other to constitute a deep architecture as shown in Fig. 1 (a).

The visible layer of the bottom-most RBM is replicated softmax (RSM) layer [7]. Unlike [7], we model count data of character uni/bi-grams. The hidden layer of the top-most RBM is linear which represents the low-dimensional codes in the projection space while intermediate layers are stochastic binary. The autoencoder is trained in two phases: *i)* greedy layer wise pre-training and, *ii)* fine-tuning through backpropagation. During pre-training, each RBM is trained using contrastive divergence (CD_1) learning for 50 epoch where CD_1 refers to CD with 1 step of alternating Gibbs sampling [8]. Once the network is pre-trained, the autoencoder is unrolled as shown in Fig. 1 (b) and the cross-entropy error between input and output is backpropagated to adjust the weights of the entire network.

As shown in Fig. 1 (a), the autoencoder is trained with original word and its transliteration together. We use the transliteration pairs training data provided during the track. After training, the autoencoder is able to project the terms from both the scripts to a low dimensional space (m=20).

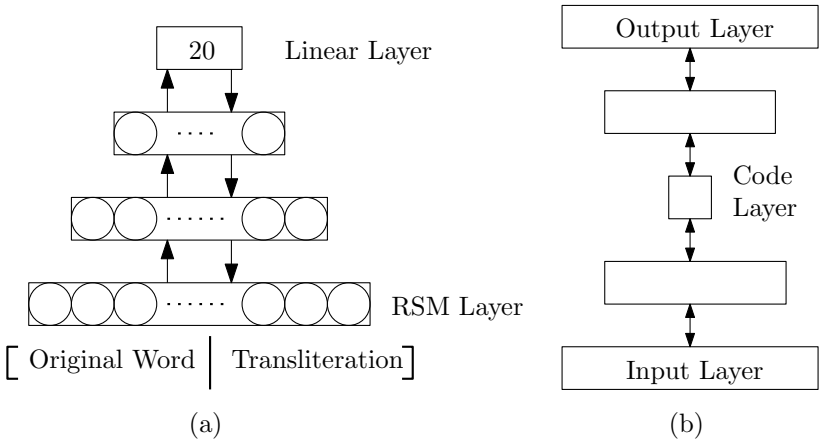

Fig. 1. The architecture of the autoencoder to learn character level variation.

3.2 Intra/Inter script Spelling Variants

We project the lexicon of the document collection in the 20-d projection space using the trained autoencoder. The given term t is also projected in the 20-d space. All the terms with $sim(\vec{t}, \vec{t_{v_i}}) > \theta$ are considered as variants of the term t where sim is $cosine$ similarity function, $\vec{t}$ and $\vec{t_{v_i}}$ are 20-d representation of t and i^{th} variant of t respectively and θ is a similarity threshold.

3.3 Subtask-1

The first subtask can be split in two parts: *i)* language identification and, *ii)* transliteration. In order to identify the language, we took the collection of Hindi terms transliterated in Roman script and proper English terms for training a support vector machine (SVM). These terms are transformed into character 1/2/3-grams as features. SVM is used as a binary classification where 0 being English term and 1 as Hindi term transliterated in Roman script.

For the transliteration, we find the nearest (Devanagari script) Hindi term from the projection space as its transliteration. It can also be seen as a transliteration mining operation.

3.4 Subtask-2

As the task is to retrieve relevant songs lyrics, the sequential information among the terms is very important. Simple bag-of-words based models would lead to very noisy retrieval. Hence we index the songs collection as word 2-grams and use word 2-gram variant of TF-IDF like models for retrieval.

Query formulation We first find the inter/intra script variants of the query terms as explained in Sec. 3.2. Then we formulate the query as word-2 grams taking account the variants in the corresponding scripts. The query formulation is explained by example in Table 1.

Original Query	ik din ayega
Variants of "ik"	"ik", "ikk", "ig", "एक", "इक"
Variants of "din"	"din", "didn", "diin", "दिन", "डिन"
Variants of "ayega"	"ayega", "aeyega", "ayegaa", "आयेगा", "आएगा"
Formulated Query[2]	ikdin, ikdidn, ik$diin, ikk$din, ikk$didn, ikk$diin, igdin, igdidn, ig$diin, din$ayega, din$aeyega, din$ayegaa, didn$ayega, didn$aeyega, didn$ayegaa, diin$ayega, diin$aeyega, diin$ayegaa, एक$दिन, एक$डिन, इक$दिन, इक$डिन, दिन$आयेगा, दिन$आएगा, डिन$आयेगा, डिन$आएगा

Table 1. Example of query formulation for transliterated search.

4 Results and Analysis

We evaluated the performance of our approach on the dataset provided under transliterated search shared task.

4.1 Subtask-1

Simple SVM based classification to identify language seems to work well. Table 2 depicts the results of the submission. It can be noticed that even with a simple classification module the labeling accuracy is around 95%. Our analysis revealed that some of the terms can be either classified as English or Hindi. Usually such terms are quite short in length *e.g. to (तो), me (मे), chain (चैन), fool (फूल)* and so on. Although such terms are not in majority but should be tackled properly.

Labeling		Transliteration	
Labeling Accuracy	0.9540	Transliteration F-Score (run-1)	0.4209
Labeling F-Score (English)	0.9019	Transliteration F-Score (run-2)	0.4311
Labeling F-Score (Hindi)	0.9700	Transliteration F-Score (run-3)	0.3796

Table 2. Results for query labeling subtask.

Our first and second runs differ in terms of reference collection used to find the transliteration. The first run considers Hindi Wikipedia while the second run uses the union of Wikipedia and lyrics collection lexicons. In third run we apply some very basic

heuristics to re-rank the transliteration but it does not help much. Being the transliteration mining approach the performance is upper bounded by the coverage of the collection. Moreover Wikipedia in Hindi is quite noisy and contains a lot of terms which are wrongly spelled. In the light of this phenomenon, misspelled terms like विवाह:, कया, इबदत, विधि, फ़रीयाद, दुइनया are mined instead of विवाह, क्या, इबादत, विधि, फरियाद, दुनिया. We believe using a more extensive and linguistically correct reference collection can drastically improve the performance in such cases which are quite a lot in our case. In general the transliteration mining algorithm is quite strong because it is able to find $\sim$43% transliteration accurately from the reference lexicon of more than 400k terms despite the possibility that many terms might not be present or correctly spelled in the collection.

4.2 Subtask-2

Our method of handling query term spelling variation leads to very strong results in terms of retrieval. The results of this subtask are presented in Table 3. The basic difference between the runs is the selection of parameter values. There are three parameters of the method: *i)* min_len - minimum term length to consider for variation look-up to expand the query, *ii)* θ - similarity threshold and, *iii)* $algo$ - ranking algorithm. For runs 1, 2 and 3 the $\{min\_len, \theta, algo\}$ configuration is $\{2, 0.95, \text{tf-idf}\}$, $\{2, 0.95, \text{XSrqA\_M}\}$ and $\{3, 0.95, \text{XSrqA\_M}\}$. XSrqA_M is an unsupervised divergence from randomness (DFR) model [9] especially suitable for short and medium length queries such as in this case.

Metric	Run-1	Run-2	Run-3	Score$_{max}$	Score$_{median}$
nDCG@5	0.7669	0.8052	0.7584	0.8052	0.5620
nDCG@10	0.7642	0.8002	0.7534	0.8002	0.5608
MAP	0.4209	0.4236	0.3558	0.4236	0.2355
MRR	0.7747	0.8440	0.7773	0.8440	0.5884

Table 3. Results of the *ad hoc* retrieval task in transliterated space. Score$_{max}$ and Score$_{median}$ are the maximum and median scores of 8 runs from 3 different teams.

The algorithm is able to find relevant documents at very high ranks. The method is a good blend of precision and recall oriented measures. The word 2-grams variant of the retrieval model ensures high precision and the relaxed θ criterion for inter/intra script variation ensures recall. It can be noticed from Table 1, some of the variants are not valid *"ig"* but when combined with next term *"din"* minimises chances of query drift.

The proposed method is quite efficient as well because it can formulate the query shown in Table 1 in less than 1 second using a single CPU thread. The proposed method can easily be parallelised with some smart matrix multiplication approaches like using GPUs.

5 Conclusions

The proposed approach can effectively handle the spelling variation across the scripts. The learning algorithm is able to learn important character-level mappings within and across the scripts jointly and does not depend on any hand crafted rules or external resource except training data of transliteration pairs. The proposed method is also efficient for large scale retrieval.

Acknowledgment

The work of the frst two authors was carried out in the framework of the WIQ-EI IRSES project (Grant No. 269180) within the FP 7 Marie Curie, the DIANA APPLICATIONS Finding Hidden Knowledge in Texts: Applications (TIN2012-38603-C02-01) project and the VLC/CAMPUS Microcluster on Multimodal Interaction in Intelligent Systems.

References

1. Saha Roy, R., Choudhury, M., Majumder, P., Agarwal, K.: Overview and Datasets of FIRE 2013 Track on Transliterated Search. In: Proceedings of the Fifth Forum for Information Retrieval Evaluation. FIRE '13, India, Information Retrieval Society of India (2013)
2. Zobel, J., Dart, P.: Phonetic string matching: lessons from information retrieval. In: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval. SIGIR '96, New York, NY, USA, ACM (1996) 166–172
3. Kumaran, A., Khapra, M.M., Li, H.: Report of news 2010 transliteration mining shared task. In: Proceedings of the 2010 Named Entities Workshop. NEWS '10, Stroudsburg, PA, USA, Association for Computational Linguistics (2010) 21–28
4. Udupa, R., Saravanan, K., Bakalov, A., Bhole, A.: "they are out there, if you know where to look": Mining transliterations of oov query terms for cross-language information retrieval. In: ECIR. (2009) 437–448
5. Udupa, R., Khapra, M.M.: Transliteration equivalence using canonical correlation analysis. In: ECIR. (2010) 75–86
6. Fukunishi, T., Finch, A., Yamamoto, S., Sumita, E.: A bayesian alignment approach to transliteration mining. **12**(3) (August 2013) 9:1–9:22
7. Salakhutdinov, R., Hinton, G.E.: Replicated softmax: an undirected topic model. In: NIPS. (2009) 1607–1614
8. Hinton, G.E.: Training products of experts by minimizing contrastive divergence. Neural Computation **14**(8) (2002) 1771–1800
9. Amati, G.: Frequentist and bayesian approach to information retrieval. In: Proceedings of the 28th European conference on Advances in Information Retrieval. ECIR'06, Berlin, Heidelberg, Springer-Verlag (2006) 13–24