Multilingual Indexing Support for CLIR using Language Modeling

Prasad Pingali, Vasudeva Varma
{pvvpr, vv}@iiit.ac.in
International Institute of Information Technology
Hyderabad, India.

Abstract

An indexing model is the heart of an Information Retrieval (IR) system. Data structures such as term based inverted indices have proved to be very effective for IR using vector space retrieval models. However, when functional aspects of such models were tested, it was soon felt that better relevance models were required to more accurately compute the relevance of a document towards a query. It was shown that language modeling approaches [1] in monolingual IR tasks improve the quality of search results in comparison with TFIDF [2] algorithm. The disadvantage of language modeling approaches when used in monolingual IR task as suggested in [1] is that they would require both the inverted index (term-to-document) and the forward index (document-to-term) to be able to compute the rank of document for a given query. This calls for an additional space and computation overhead when compared to inverted index models. Such a cost may be acceptable if the quality of search results are significantly improved. In a Cross-lingual IR (CLIR) task, we have previously shown in [3] that using a bilingual dictionary along with term co-occurrence statistics and language modeling approach helps improve the functional IR performance. However, no studies exist on the performance overhead in a CLIR task due to language modeling. In this paper we present an augmented index model which can be used for fast retrieval while having the benefits of language modeling in a CLIR task. The model is capable of retrieval and ranking with or without query expansion techniques using term collocation statistics of the indexed corpus. Finally we conduct performance related experiments on our indexing model to determine the cost overheads on space and time.

1 Introduction

Information retrieval (IR) is the science of searching for information in documents, searching for documents themselves, searching for metadata which describe documents, or searching within databases, whether relational stand-alone databases or hypertext networked databases such as the Internet or World Wide Web or intranets, for text, sound, images or data. Searching for the relevant information also may involve the notion of a ranking function, which denotes the relevance of a retrieved item for the given query. It should be noted that ranking the search results in the order of their relevance to query is an important aspect in most of the IR systems that

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

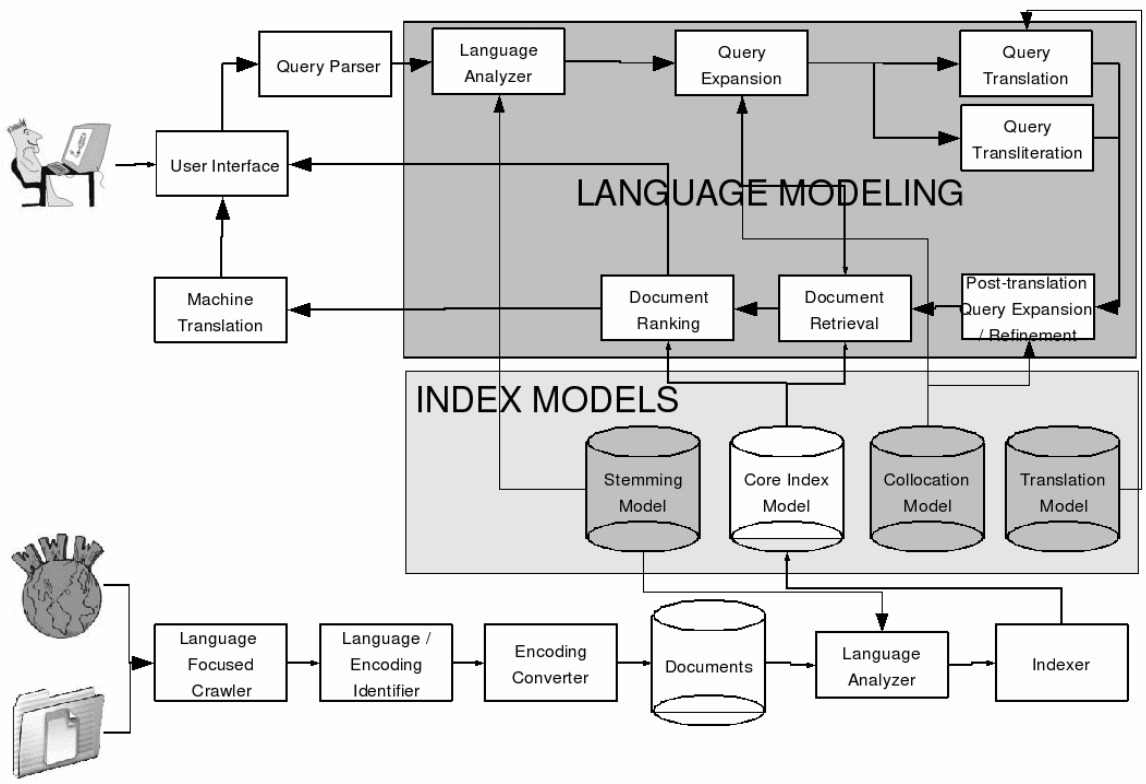

Figure 1: A Typical Information Access Framework

deal with text document retrieval. Boolean IR systems are an exception to this observation, where all the search results that satisfy the boolean constraints of the query are presented to the user in the same order as found in the database. However, boolean retrieval systems were found to be insufficient in building usable applications in many domains. In this paper we deal with the problem of text document retrieval and more specifically Cross-lingual Information Retrieval (CLIR).

CLIR has been an important area of research in the IR community and the need is felt to extend today's monolingual techniques to be able to simultaneously handle content of multiple languages. Assume a query Q_s in user's language (also known as a source language). The core cross-lingual IR problem is defined as retrieving and ranking of relevant documents which may occur in the same source language as that of Q_s and each of a set of target languages $TL_1, TL_2....TL_k$. This problem can be first viewed in terms of a single target language (say the cross language document collection is D_t) and then scaled to multiple languages using the same technique. However, such a solution will not try to exploit the availability of same or similar (comparable) information in multiple languages. In CLIR and Multi-lingual IR (MLIR) research, studies exist which address the translingual IR problem by considering a single target language [4–7], and also by exploiting similar information in multiple target languages [8, 9]. In this study we consider solutions for one target language and then scale such a solution for other languages. This would enable us *not* to assume any comparable data or resources available in multiple target languages and at the same we would not try to do anything special in the ranking function when multiple target languages exist. If both the query and documents were in the same language, all the standard IR techniques (such as weighted keywords, vector space models, probabilistic IR models) can be directly applied to retrieve and rank the documents with slight adaptations to Indian language content. However these techniques may not be directly applicable for the cross-language retrieval problem. In order to achieve the CLIR, some of the possible approaches could be:

- *Translate the document collection* - Manually or automatically translate all the documents in D_t into the source language document set, say $D_s^{'}$ and retrieve and rank the documents using the same language query Q_s using any of the monolingual IR techniques

- *Translate the query* - Convert the source language query Q_s into the target language either manually, automatically or semi-automatically with user interaction (say as $Q_t^{'}$) and retrieve and rank the documents present in the same language (i.e. D_t) using any of the monolingual IR techniques

- *cross-lingual relevance feedback* - If a parallel corpus exists between the query and document languages, i.e., for every source language document in D_s there exists an equivalent target language translated document in D_t which is identifiably annotated. In such a setup, the source language query Q_s can be used to query documents of the same language first, and the retrieved documents from source language are used to obtain their equivalent target language documents from D_t. The terms collected from these documents are then used to retrieve target language documents.

While these different approaches have been studied to address CLIR problem, each of them has its own disadvantages. Translating the entire document collection might work if the document collection is static and is locally available. However these may be unrealistic assumptions if a CLIR system were to be built for an environment like web where the document collection is very huge and is always changing at a very rapid rate. On the other hand translating the query seems promising, but queries are typically too short to provide enough context to precisely translate into a different language. Despite this drawback, CLIR researchers have argued and demonstrated that a precise translation may not be required, since the translated query is not for human consumption and is meant to obtain the gist of the information need. The third approach of relevance feedback has been shown to perform very well [6] but it requires a parallel corpus which is a difficult resource to obtain for many language pairs. Since we are looking at Indian languages as the query language, it is difficult to find any such resources. Therefore we limit our scope to using dictionary based translation of the query, with query expansion using monolingual corpus (i.e. the second approach).

2 System Overview

Having defined the cross-lingual IR problem in general, let us now look at each of the sub-problems to achieve the stated larger goal. This understanding is essential to appreciate some of the decisions made in designing the structure of the index. In order to illustrate some of the major problem areas within CLIR we provide a typical system overview as shown in Figure 1 and use an example information need represented as a query. Consider a user trying to locate documents discussing about various "Nestle's products". This query would be written as "नेस्ले के उत्पादन" in Hindi and as "నెస్లే యొక్క ఉత్పత్తులు " in Telugu [1]. As evident from the Figure 1, there are two broad logical divisions in this CLIR sub-problems. The first logical division consists of offline processing modules to collect and index the document collections. The second logical division deals with runtime processing of query processing, retrieval, ranking and presentation of information to the user. Some of the modules in the shown in Figure 1 are not in the scope of this paper and hence will not be discussed in detail. Let us now look at the functions of various sub-problems or modules that are part of the above mentioned framework that would be dependent on the underlying design of database. The following sections 2.1 to 2.4 discuss offline processing mainly involving data collection and indexing. Sections 2.5 to 2.10 discuss online processing of the IR system at the time of issue of a query by the user.

[1] Hindi and Telugu are two major Indian languages geographically corresponding to Northern and South-Eastern India, respectively.

2.1 Language / Encoding Identifier

As mentioned in the previous example query, our system is expected to accept queries from multiple Indian languages. Similarly, the document collections that are to be retrieved can be present in multiple languages and multiple encodings in each language. A number of language encodings are in usage on the world wide web and other document repositories. Especially in Indian language document repositories, these document collections are available in a set of national and international standards for Indian language character set encodings, while a number of publishers use proprietary non-standard encodings. In order to be able to index such content, it is very important to identify the language and such encodings in order to achieve a better recall in retrieval by having much broader coverage. For language identification a set of statistical and non-statistical (rule based) techniques can be used to identify the language of a document with a certain precision. Script based recognition for Unicode character sets works to an extent, but still ambiguities might exist. For example, while most of the major Indian languages have a script of their own, Marathi, Hindi, Nepali and Sanskrit all occur in the Devanagari script, in which case script based heuristics may not work. Therefore for ambiguous scripts a better set of heuristics are required or statistical language identification techniques can help. In our work, we use a set of heuristic based techniques using fonts and character ranges to recognize character encodings such as UTF-8, ISCII or other proprietary encodings from HTML and PDF documents [10].

2.2 Encoding Converter

Once languages and character encodings are identified for a particular document or a particular piece of text within a document, such content needs to be converted into a standard encoding so that it can be indexed and retrieved. Unless all the character encodings of a single language are converted into a single encoding, it will not be possible to compare strings and match queries for retrieval. Therefore an automatic encoding converter is essential to enable information access. Again similar to language identification, a set of heuristic based, or statistical methods can be used to convert content of one encoding into another. For example, one could come up with a mapping table of various glyphs or glyph sequences of one encoding into another which can be then used to automatically convert one character encoding into another. Since Indian languages are syllabic in nature most of these are rarely one to one glyph mappings and end up being one to many or many to one. An example implementation of such encoding conversion into UTF-8 is discussed in [10].

2.3 Language Analyzer

A language analyzer program performs a set of natural language tasks on the indexable content and queries before passing it to the indexer or retrieval engine. Some of the standard and broad tasks of a language analyzer w.r.t IR would be to tokenize the input text and identify valid tokens of language, identify which of the valid tokens are worth indexing and which are not (also known as stop word identification) and performing some level of morphological processing on these tokens such as stemming. For instance, the example queries of "नेस्ले के उत्पादन" and "నెస్లే మొక్క ఉత్పత్తులు " may be converted into "नेस्ले उत्पाद" and "నెస్లే ఉత్ప " after stop-word elimination and stemming. In order to perform these tasks, a language analyzer specific to each language may be needed to be built or on the other hand a generic solution can be devised which may work for all the languages. Tokenization of string into tokens is a fairly standard heuristic based process, however stop word identification and stemming may have to be language specific if heuristics or word lists were used [11, 12]. Some research has been done on this problem using statistical stemming techniques [13, 14].

2.4 Indexer

An indexer program builds an index database. This index could be an inverted index or some other data structure suitable for IR with some meta-data useful for ranking and presentation of information to the user. Each of

the indexable tokens obtained after stop-word removal and stemming are used by the indexer module to store information such as the documents in which the token has occurred and frequencies and position of such occurrences in these documents etc. Advanced indexers also have capabilities to store fielded indices. A fielded index would store the index information per field per document. For example a search engine application may want to provide ability to search a particular portion or meta-data of a document, such as the title of the document or the body text or web-based fields such as search within a given website etc. An indexer module should be capable of building data structures which enable fast retrieval of relevant documents and also enable ranking of these documents. The design of data-structure for index should also be sensitive to transaction aspects such as locking of indices for writes, updatability of index without difficulty etc.

2.5 Query Expansion

A query expansion module is an optional module for a search engine, which is used to add/rephrase/refine some keywords to the query. A query is viewed as user's expression of information need in the form of a set of keywords. For the user to express information need as keywords, the user has to guess the language model (keywords) occurring in the documents relevant to his/her information n.eed. Usually the users may not be able to guess the right keywords that may be found in the relevant documents thereby resulting in a poor user satisfaction. A query expansion module would automatically try to add more keywords to the user's query resulting in better user satisfaction. Query expansion could be achieved by adding synonymous words or semantically related words from a thesaurus or by using collocation statistics from a monolingual corpus. For example, a few terms can be added to the example query previous mentioned using co-occurrence statistics to add terms such as "मिल्कमेड , मेगि"(Milkmaid, Maggi) which co-occur with "नेस्ले" (Nestle) in a large Hindi corpus.

2.6 Query Translation

A query translation module is required to achieve CLIR. Query translation can be achieved using a bilingual lexicon or translation models built using parallel corpus. Bilingual lexicon can be obtained by digitizing and converting human readable dictionaries or can be obtained by manually building them with the help of linguists. Two types of bilingual lexica exist, one which just lists all the possible translations for a given source language word, the second one which not only lists the possible translations, but also comes with some meta-data for the translation, such as the synonyms, the probability of translation etc. Lexicon with translation probabilities are sometimes referred to as statistical lexicon. Statistical lexicon can be automatically built using parallel corpora. In some cases the direction of translation may be different from the direction of the CLIR task. For example, bilingual lexicon may provide translations for words of language L_1 into L_2, while the CLIR system might take in queries of L_2 to retrieve documents of L_1. In such situations proper set of heuristics need to be used in order to translate queries using a reverse lookup of the dictionary. In case of a statistical lexicon, a likelihood estimate for the translations needs to be computed. In the previous example query, "उत्पाद" would get translated as "product, produce, production, producer" etc., since we store only the stems and hence translate stems. Storing the complete words may result in higher translation precision, but would lead to lower recall since most of the word formations may not be found in the dictionary.

2.7 Query Transliteration

Not all queries can be translated into the target language using a bilingual lexicon. One of the important issues for query translation in CLIR systems is the handling of out of vocabulary words (or OOVs). Words such as proper nouns, words borrowed from a foreign language fall under this category. Such words are directly transliterated into the script of the target language. Such transliterations are not always exact transliterations and

result in being influenced by the dialect and morphology of the target language. It is very common to find such keywords in the queries of CLIR systems where the source and target languages share a high number of cognates. Handling OOVs for CLIR systems across various Indian languages becomes very important since various Indian languages share higher proportion of cognates when compared with English. Heuristic based approaches using phoneme mappings or probabilistic transliteration models [15] can be used to address the problem of OOVs. This module is responsible to convert "नेस्ले" (written as Neslay in Roman script) and "నెస్లే " (written as Neslay in Roman script) as "Nestle".

2.8 Post-translation Query Expansion/Refinement

We differentiate post-translation query expansion from the previous query expansion module, since this module not only adds more keywords to the translated query, but also performs query refinement by eliminating noisy translations from the previous modules. Query refinement can be achieved by disambiguation from any context information during the query translation process or using techniques such as pseudo relevance feedback.

2.9 Document Retrieval

Document retrieval forms the central problem of IR. The problem of document retrieval is to retrieve documents relevant to the user's information need from a given document collection. The concept of relevance is very difficult to define. In naive terms, all the documents containing the keywords from the user's queries may be treated as relevant, however, such a definition will always have exceptions. A number of relevance definitions exist which not only are a function of queries and documents but a number of other parameters such as the user and the context in which the query was issued. However, most of the IR research ignores these other dimensions and largely tries to define relevance as a function of user's query and the documents from which a relevant subset needs to be identified.

2.10 Document Ranking

Document ranking problem is a function performed after retrieval of relevant documents. The goal of document ranking module is to find the best way to order the set of retrieved documents such that the items/documents are ordered in the decreasing order of their relevance. In many IR models ranking is achieved using the same relevance function as mentioned in the previous module. However in some relevance models such as boolean IR model [16] such a ranking is not inherent in the relevance function.

While each of the above mentioned sub-problems in a typical CLIR framework is a research worthy problem in itself, a set of frameworks / solutions can be defined in such a way that they solve more than one of the above sub-problems. Such frameworks could be, for example, a vector space framework, probabilistic framework or ontology based frameworks. In other words this entire problem of information retrieval can be viewed in two dimensions. One is the vertical dimension which is that of each of the sub-problems mentioned above. The other is a horizontal dimension of an approach or framework within which generalized solutions to an extent can be achieved which address more than one of the above mentioned sub-problems.

Traditionally, the language modeling technique for IR as described in [1] addresses only document retrieval and ranking modules. We extend the scope of our problem to also include query translation and query expansion modules.

3 Problem Definition

The problem being addressed in this paper is to design an underlying indexing mechanism that can support easy retrieval and ranking of documents for a cross-lingual query using the language modeling based ranking

functions mentioned in this section. Before looking at the indexing solution, we need to look at the document retrieval and ranking functions in order to understand the computation that is required during the retrieval time for a cross-lingual query. This would in turn lead to the requirements for the indexing module.

We propose a language modeling (horizontal) approach to CLIR as shown in Figure 1 which cuts across a number of information access sub-problems (verticals). Statistical language models are probability distributions defined on sequences of elementary units, $P(u_1...u_n)$. Language modeling has been used in many NLP applications such as part-of-speech tagging, parsing, speech recognition, machine translation and information retrieval. Estimating sequences can become expensive in corpora where phrases or sentences can be arbitrarily long (data sparseness problem), and so these models are most often approximated using smoothed N-gram models based on unigrams, bigrams and/or trigrams.

In speech recognition and in most of the other applications of language modeling, these models refer to a probabilistic distribution capturing the statistics of the generation of a language, and attempt to predict the next word in a speech or text or state sequence. However the language models used in information retrieval may be viewed in a slightly different way. When used in information retrieval, a language model is associated with a document in a collection. With query Q as input, retrieved documents are ranked based on the probability that the document's language model (M_d) would generate the terms of the query, $P(Q|M_d)$. And such a language need not be sensitive to term sequence most of the times, since IR applications typically assume bag-of-words model for documents and queries.

The elementary unit in our IR models is a term. We define a term to be a sequence of characters which are either words or conflated words (stems). We propose to build and apply term based language models to various CLIR functions such as query enrichment, query translation, document retrieval and ranking. The language models being proposed here are trying to model semantically related words and unigram language models rather than actually modelling the sequences of terms as in other NLP applications such as speech recognition or part-of-speech tagging. In next few sub-sections we present term based language models which need to be supported by the underlying indexing mechanism.

3.1 Query Expansion

To achieve query expansion, it is required to automatically predict the keywords in the relevant documents to the given information need of the user. In order to achieve this in the absence of any further user context, one can add semantically related keywords to the query to enable retrieval of relevant documents with high precision and recall. Motivated by the fact that the meaning of a new concept can be learned from its usage with other concepts within the same concept [17], we automatically compute the dependencies of a word w on other words based on their lexical co-occurrence in the context of w in a sufficiently large corpus. A term collocation matrix is constructed by taking a window of length k words and moving it across the corpus at one term increments. All words in the window are said to co-occur with the first word with a given weight. Such a weight can be determined to be a function of the distance between the co-occurring words or can be assumed to be of equal weight. The weights assigned to each co-occurrence of terms are accumulated over the entire corpus. That is, if $n(w, k, w')$ denote the number of times word w' occurs k distance away from w when considered in a window of length K, then

$$\mathrm{P}(w', w) \;=\; \frac{\sum_{k=0}^{K} n(w, k, w')}{|C|}$$

where $|C|$ is the size of the corpus. This equation is equal to bigram probability distribution if the window length $K = 1$.

Once such term-by-term language models are built, it becomes very easy to expand given queries or documents using such language models.

3.2 Query Translation

Query translation functionality for the purpose of cross-language retrieval can be achieved in multiple ways. In this paper our focus is on using bilingual lexicon for this task and we describe how a bilingual lexicon can be used to build a probabilistic component which can be embedded into a retrieval model. The embedding of this translation component becomes easier since we assume the process of translation is independent of the actual retrieval and ranking and we perform translations at the term level.

We define translation probability of a target language term t_k from source language term s_i as $P(t_k|s_i)$. This model works well for CLIR since the system need not zero down onto a single possible translation for a given query. Therefore we compute the translation distribution for a given query and only refine the noisy translations with whatever evidence is available. In essence, this probabilistic translation model results in a probabilistic graphical model, where each source language term can get translated to multiple target language terms with different probabilities. This essentially gives us the graphical structure of dependencies (source/target dependencies) and the conditional probability distribution which constitutes our model.

Calculation of such a model can be again achieved in different ways. Ideally, the use of a bilingual parallel corpus would provide the best way to estimate the conditional probabilities and enables building of a statistical bilingual lexicon. However in [18] it was shown that, even in the absence of a bilingual parallel corpus, a conditional probability distribution can be achieved by assuming uniform probabilities of all possible translations to a given term. Apart from these we also propose assuming some underlying distribution based on the position of a given meaning in the lexicon, such as picking only one translation or assume the ordering of translations in the lexicon to bear an importance. Therefore, the query translation model should be capable of handling all possible term translations with weights associated with the translation.

3.3 Document Retrieval and Ranking

In [1], a language modeling framework was defined to retrieve and rank documents for the given information needs (queries). The model is non-parametric and does not assume any underlying classes such as relevance or irrelevance for the items to be retrieved. Every document/item is retrieved by computing the probability of its language model emitting the given query.

Therefore, the ranking function $R(Q, d)$ is defined as

$$\begin{aligned} R(Q,d) &= P(Q|M_d) \\ &= \prod_{w_j \in Q} P(w_j|M_d) \cdot \prod_{w_j \notin Q} (1 - P(w_j|M_d)) \end{aligned} \tag{1}$$

where, Q is the user's query containing a sequence of terms q_i and M_d is the document's language model which can emit a set of terms w_j.

Inspired by this model we extend this model to serve the various functions of CLIR. To include query expansion as part of the retrieval and ranking function, we obtain an expanded query $Q^{'}$ and rewrite the ranking function as

$$\begin{aligned} R(Q,d) &= P(Q^{'}|Q).P(Q^{'}|M_d) \\ &= \prod_{w_j \in Q^{'}} P(w_j, Q).P(w_j|M_d) \cdot \prod_{w_j \notin Q^{'}} (1 - P(w_j|M_d)) \end{aligned} \tag{2}$$

where $P(w_j, Q)$ gives the weight of the expanded term w_j in the context of the given query Q. Similarly query translation probabilities can be included as part of the ranking function, where $Q^{'}$ becomes the translated query or translated and expanded query as the sequence of the modules are plugged in. The actual ranking of

documents happens on the probability that the document's language model emits the transformed query, while accounting for the joint probability of the transformation itself.

4 Multilingual Index Implementation

Given the language modeling framework described in the previous section, our task is to design an indexing mechanism to support such a CLIR system. For this purpose, we use a traditional inverted index concept and see how it can be extended for this task. We use Lucene's[2] inverted index mechanism and modify it to suit our model. Before analyzing Lucene's index file structure, we should understand the inverted index concept. An inverted index is an inside-out arrangement of documents in which terms take center stage. Each term points to a list of documents that contain it. On the contrary, in a forward index, documents take the center stage, and each document refers to a list of terms it contains. You can use an inverted index to easily find which documents contain certain terms. Lucene uses an inverted index as its index structure while a forward index facility also exists which can be optionally created. From Equations 2 and 3 it can be observed that, not only the query terms, but also all the terms contained in a document are required to be accessible while computing the rank of a given document. Therefore we will be using both inverted and forward index as our core index model as depicted in the Figure 1. The description of this Lucene core index model is given in the Section 4.1, followed by our modifications to it in Section 4.4.

4.1 Lucene Index Structure Overview

The fundamental concepts in Lucene are index, segments, document, field and term.

An *index* contains a sequence of *segments* which contain documents. Each *document* is a sequence of fields. A *field* is a named sequence of terms and a *term* is a string. The same string in two different fields is considered a different term. Thus terms are represented as a pair of strings, the first naming the field, and the second naming text within the field.

Segments: Lucene indexes may be composed of multiple sub-indexes, or segments. Each segment is a fully independent index, which could be searched separately. Indexes evolve by creating new segments for newly added documents or by merging existing segments. Searches may involve multiple segments and/or multiple indexes, each index potentially composed of a set of segments.

Each segment index maintains the following:

- *Field names*. This contains the set of field names used in the index.
- *Stored Field values*. This contains, for each document, a list of attribute-value pairs, where the attributes are field names. These are used to store auxiliary information about the document, such as its title, url, or an identifier to access a database. The set of stored fields are what is returned for each hit when searching. This is keyed by document number.
- *Term dictionary*. A dictionary containing all of the terms used in all of the indexed fields of all of the documents. The dictionary also contains the number of documents which contain the term, and pointers to the term's frequency and proximity data.
- *Term Frequency data*. For each term in the dictionary, the numbers of all the documents that contain that term, and the frequency of the term in that document.
- *Term Proximity data*. For each term in the dictionary, the positions that the term occurs in each document.

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

Value	First byte	Second byte	Third byte
0	00000000		
1	00000001		
2	00000010		
...			
127	01111111		
128	10000000	00000001	
129	10000001	00000001	
130	10000010	00000001	
...			
16,383	11111111	01111111	
16,384	10000000	10000000	00000001
16,385	10000001	10000000	00000001
...			

Table 1: VInt Encoding Example

- *Normalization factors*. For each field in each document, a value is stored that is multiplied into the score for hits on that field.
- *Term Vectors*. For each field in each document, the term vector (sometimes called document vector or the forward index) is stored. A term vector consists of term text and term frequency.
- *Deleted documents*. An optional file indicating which documents are deleted.

4.2 Primitive Data Types

Lucene uses *Byte*, *UInt32* (4 bytes), *UInt64* (8 bytes), *VInt* and *Chars* as primitive data types. The *UInt* data types are used based on the address space requirements of 32-bits or 64-bits. A variable-length format for positive integers (*VInt*) is defined where the high-order bit of each byte indicates whether more bytes remain to be read. The low-order seven bits are appended as increasingly more significant bits in the resulting integer value. Thus values from zero to 127 may be stored in a single byte, values from 128 to 16,383 may be stored in two bytes, and so on.

It can be observed from Table 1 that the *VInt* datatype provides compression while still being efficient to decode, by using only required number of bytes to encode an integer address. Lucene writes unicode character sequences using Java's "modified UTF-8 encoding". A complete string is written as a *VInt* representing the length, followed by the actual character data.

4.3 Lucene Index Storage

The following describes the main index files in Lucene. Some of the file structures described below might not include all of the columns, but it won't affect the reader's understanding of the index file.

Segments file: A single file contains the active segments information for each index. This file lists the segments by name, and it contains the size of each segment. Table 2 describes the structure of this file.

Fields information file: Documents in the index are composed of fields, and this file contains the fields information in the segment. Tables 3 shows the master field structure. The fields master file has two slave files, namely field index file (structure described in Table 4) and field data file (described in Table 5).

Fields are numbered by their order in this file. Thus field zero is the first field in the file, field one the next, and so on. Note that, like document numbers, field numbers are segment relative.

Column name	Data type	Description
Version	UInt64	Contains the version information of the index files.
SegCount	UInt32	The number of segments in the index.
NameCounter	UInt32	Generates names for new segment files.
SegName	String	The name of one segment. If the index contains more than one segment, this column will appear more than once.
SegSize	UInt32	The size of one segment. If the index contains more than one segment, this column will appear more than once.

Table 2: Structure of Segments file

Column name	Data type	Description
FieldsCount	VInt	The number of fields.
FieldName	String	The name of one field.
FieldBits	Byte	Contains various flags. For example, if the lowest bit is 1, it means this is an indexed field; if 0, it's a non-indexed field. The second lowest-order bit is one for fields that have term vectors stored, and zero for fields without term vectors.

Table 3: Structure of Fields information file

Column name	Data type	Description
FieldValuesPosition	UInt64	This is used to find the location within the field data file of the fields of a particular document. Because it contains fixed-length data, this file may be easily randomly accessed. The position of document n's field data is the Uint64 at n*8 in this file.

Table 4: Structure of Fields Index file

Column name	Data type	Description
FieldCount	VInt	
FieldNum	VInt	
Bits	Byte	Only the low-order bit of Bits is used. It is one for tokenized fields, and zero for non-tokenized fields.
Value	String	

Table 5: Structure of Fields Data file

Column name	Data type	Description
TIVersion	UInt32	Names the version of this file's format.
TermCount	UInt64	The number of terms in this segment.
Term	Structure	This column is composed of three subcolumns: PrefixLength, Suffix, and FieldNum. It represents the contents in this term.
DocFreq	VInt	The number of documents that contain the term.
FreqDelta	VInt	Points to the frequency file.
ProxDelta	VInt	Points to the position file.

Table 6: Structure of Term information file

Column name	Data type	Description
DocDelta	VInt	It determines both the document number and term frequency. If the value is odd, the term frequency is 1; otherwise, the Freq column determines the term frequency.
Freq	VInt	If the value of DocDelta is even, this column determines the term frequency.

Table 7: Structure of the Frequency file

Column name	Data type	Description
PositionDelta	VInt	The position at which each term occurs within the documents

Table 8: Structure of the Position file

Column name	Data type	Description
TermDelta	VInt	It determines both the term number and joint frequency. If the value is odd, the joint frequency is 1; otherwise, the Freq column determines the joint frequency.
Freq	VInt	If the value of TermDelta is even, this column determines the joint frequency.

Table 9: Structure of the Collocation file

Column name	Data type	Description
TermDelta	VInt	It determines both the term number and conditional probability of translation.
TransProbability	VInt	

Table 10: Structure of the Term Translation file per language pair

Stored fields are represented by two files, one is field index and the other is field data.

Text information file: This core index file stores all of the terms and related information in the index, sorted by term. Table 6 shows the structure of this file. The pointers to term expansion file are stored in this file.

This file is sorted by Term. Terms are ordered first lexicographically by the term's field name, and within that lexicographically by the term's text.

Term text prefixes are shared. The PrefixLength is the number of initial characters from the previous term which must be pre-pended to a term's suffix in order to form the term's text. Thus, if the previous term's text was "bone" and the term is "boy", the PrefixLength is two and the suffix is "y".

Frequency file: This file contains the list of documents that contain the terms, along with the term frequency in each document. If Lucene finds a term that matches the search word in the term information file, it will visit the list in the frequency file to find which documents contain the term. Table 7 shows the primary fields of this file. TermFreq entries are ordered by increasing document number.

DocDelta determines both the document number and the frequency. In particular, DocDelta/2 is the difference between this document number and the previous document number (or zero when this is the first document in a TermFreqs). When DocDelta is odd, the frequency is one. When DocDelta is even, the frequency is read as another VInt.

For example, the TermFreqs for a term which occurs once in document seven and three times in document eleven would be the following sequence of VInts: 15, 10, 3.

Position file: This file contains the list of positions at which the term occurs within each document. You can use this information to rank the search results. Table 8 shows the structure of this file.

For example, the TermPositions for a term which occurs as the fourth term in one document, and as the fifth and ninth term in a subsequent document, would be the following sequence of VInts: 4, 5, 4

4.4 Additional Meta-Index Files

We create three additional meta-index files, namely a collocation file to store co-occurrence frequencies of two terms (structure described in Table 9), a query translation file to store weighted term-based translations between languages (shown in Table 10) and a modified text information file to provide pointers to the first two meta-

Column name	Data type	Description
TIVersion	UInt32	Names the version of this file's format.
TermCount	UInt64	The number of terms in this segment.
Term	Structure	This column is composed of three subcolumns: PrefixLength, Suffix, SuffixLength and FieldNum. It represents the contents in this term.
DocFreq	VInt	The number of documents that contain the term.
FreqDelta	VInt	Points to the frequency file.
ProxDelta	VInt	Points to the position file.
CollocationDelta	VInt	Points to the collocation file.

Table 11: Structure of modified Term information file

index files. The text information file mentioned in the core Lucene index model is overridden to accommodate CLIR functions such as stemming and query translation. We first of all modify the term structure to include the SuffixLength to determine the length of the suffix that needs to be stemmed. Storing the suffix as part of the term data structure instead of actually indexing the conflated variants allows the CLIR system to be able to search with or without stemming at the same time. These modifications to the term datastructure can be seen from Table 11. Having customized the underlying index structure, we conducted CLIR experiments for Indian language - English language pairs which are discussed in the next section.

5 Index Performance Evaluation

The modified Lucene indexer was evaluated for indexing time and retrieval time. It is compared against the core Lucene index model which forms our baseline. The evaluations were conducted for 3 runs for both core Lucene model and language model based index. An average of the 3-runs is being reported. Table 12 shows the times taken by the indexer to index 100,000 documents belonging to Hindi and Telugu in UTF-8 format. For the baseline run we used the Lucene's standard analyzer by modifying it to accept UTF-8 content, while for the modified Lucene index we used our own language analyzer which has an Indian language rule based stemmer. In both these runs the JVM was given a runtime memory of 1GB. Similarly, Table 13 shows the times taken for document retrieval for both the runs. We issued a set of 25 Hindi and 25 Telugu queries in a sequence with a single thread and show the average time for retrieval and ranking. In the case of monolingual retrieval runs, only same language documents are retrieved while for cross-lingual run, the queries were also translated into the other language and both the language documents were retrieved. The results of our indexer with and without query expansion module for monolingual and cross-lingual tasks are being reported in Table 13.

5.1 Evaluation Setup

An Intel Xeon 3.0 GHz machine with 4GB RAM and an IDE disk with 7,200 RPM speed was used for our experiments. We used Lucene's version 1.4 as our baseline and extended it as described in Section 4.4. The programming was done using Sun's Java 1.4.2 on a Fedora Core 3 operating system. We indexed a set of 50,000 Hindi and 50,000 Telugu news article HTML documents which were encoded in UTF-8. The total size of the document collection was approximately 700MB, with an average file size of 7.14KB.

We also performed a load-test on our language modeling based indexer for monolingual retrieval, by simulating load with 200, 300, and 400 simultaneous users with a ramp-up time of 1 second. It can be seen from Figure 2 that the index is able to provide a good throughput. The load test for 200 users showed 0% drop rate[3], while the drop rates for 300 and 400 users were 1% and 2.75% respectively.

It can be observed from Tables 12 and 13 that the language modeling based index is slower than the ordinary TF.IDF based indexing model both during indexing and retrieval. However, functionally the language modeling based index provides superior search results than the vector space model both for monolingual retrieval [1] as well as cross-lingual retrieval [3]. This improvement is even higher in languages where resources are difficult to obtain. To give a perspective of the kind of functional improvement, we present the evaluation results of the experiments mentioned in [3] in Table 14. These experiments were conducted using CLEF 2006[4] dataset for CLIR evaluation of Hindi-English and Telugu-English tasks where the queries are in Hindi, Telugu and the information is in English. HNTD and TETD runs represent the baseline experiments using the TF.IDF based retrieval using dictionary based translation, while HNLM and TELM runs represent language modeling based retrieval for Hindi and Telugu queries respectively. Measures such as Mean Average Precision (MAP),

[3] A HTTP request which could not be served due to load is treated as a dropped request. *Drop rate* determines the frequency of failure of the system for a given load.

[4] Cross Language Evaluation Forum. http://www.clef-campaign.org

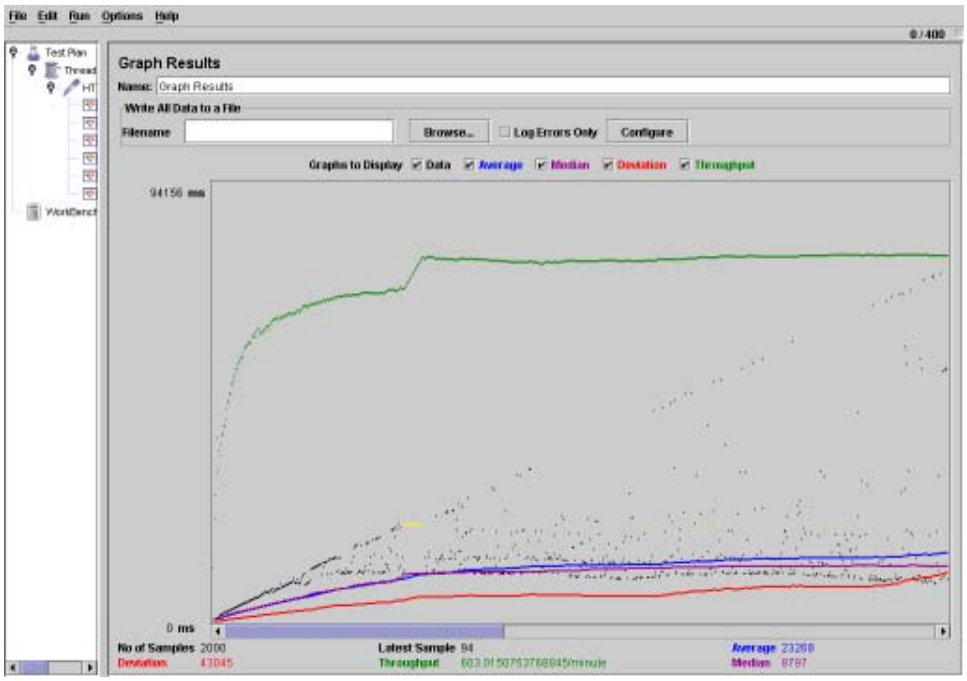

Figure 2: Load Testing on Our Index Model using 300 simultaneous users with a rampup of 1 second

Run	Index time/1000 docs	Memory
Baseline	89 sec	1024MB given to JVM
Lang. Model	359.71 sec	1024MB given to JVM

Table 12: Indexing Statistics

R-Precision and Geometric Average Precision (GAP) clearly show significant improvement. The performances of HNLM and TELM runs are comparable with the top performing systems at CLEF 2006.

6 Conclusion

In this paper we presented an indexing model for cross language information retrieval using language modeling technique. We used Lucene's inverted index model as our base model and extended it to support language modeling based CLIR. We conducted a set of experiments to test the performance of the new index model during index time as well as retrieval time. We found that consistently the new index model takes longer time to index and retrieve documents when compared to the vector space models. However, it has been shown in information

Run	Retrieval type	Avg. Retrieval Time/Query
Baseline	monolingual	667ms
LM	monolingual	941ms
LM + query expansion	monolingual	1009ms
LM	cross-lingual	1156ms
LM + query expansion	cross-lingual	1521ms

Table 13: Retrieval Statistics

Run-Id	total Relevant	Relevant-Retrieved	MAP (%)	R-Prec.(%)	GAP (%)	B-Pref.(%)
HNTD	1,258	650	12.52	13.16	2.41	10.91
TETD	1,258	554	8.16	8.42	0.36	7.84
HNLM	1,258	1051	26.82	26.33	9.41	25.19
TELM	1,258	1018	23.72	25.50	9.17	24.35

Table 14: Summary of average results for various CLIR runs

retrieval research that the language modeling based techniques have shown considerable improvement in the quality of search results provided in comparison to TFIDF algorithm. Therefore, we conclude that while there is an increase in time taken by the new index, it can provide better search results when compared to the traditional vector space based IR models.

Acknowledgment

We would like to thank the Department of Science and Technology, Government of India for partially funding this research.

References

[1] J. M. Ponte and W. B. Croft, "A language modeling approach to information retrieval," in *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM Press, 1998, pp. 275–281.

[2] G. Salton and C. Buckley, "Term-weighting Approaches in Automatic Text Retrieval," *Information Process. Management*, vol. 24, no. 5, pp. 513–523, 1988.

[3] P. Pingali, K. K. Tune, and V. Varma, "Hindi, Telugu, Oromo, English CLIR Evaluation," *Lecture Notes in Computer Science: CLEF 2006 Proceedings*, 2007.

[4] L. Ballesteros and W. B. Croft, "Resolving ambiguity for cross-language retrieval," in *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM Press, 1998, pp. 64–71.

[5] W. B. Croft and L. Ballesteros, "Phrasal translation and query expansion techniques for cross-language information retrieval," in *SIGIR '97: Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM Press, 1997, pp. 84–91.

[6] P. Clough and M. Sanderson, "Measuring pseudo relevance feedback & clir," in *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM Press, 2004, pp. 484–485.

[7] D. He, D. W. Oard, J. Wang, J. Luo, D. Demner-Fushman, K. Darwish, P. Resnik, S. Khudanpur, M. Nossal, M. Subotin, and A. Leuski, "Making miracles: Interactive translingual search for cebuano and hindi," *ACM Transactions on Asian Language Information Processing (TALIP)*, vol. 2, no. 3, pp. 219–244, 2003.

[8] L. Ballesteros and M. Sanderson, "Addressing the lack of direct translation resources for cross-language retrieval," in *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*. New York, NY, USA: ACM Press, 2003, pp. 147–152.

[9] J. Mayfield and P. McNamee, "Triangulation without translation," in *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM Press, 2004, pp. 490–491.

[10] P. Pingali, J. Jagarlamudi, and V. Varma, "Webkhoj: Indian language ir from multiple character encodings," in *WWW '06: Proceedings of the 15th international conference on World Wide Web*. Edinburgh, Scotland: ACM Press, 2006, pp. 801–809.

[11] L. S. Larkey, L. Ballesteros, and M. E. Connell, "Improving stemming for arabic information retrieval: light stemming and co-occurrence analysis," in *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM Press, 2002, pp. 275–282.

[12] C. Fox, "A stop list for general text," *SIGIR Forum*, vol. 24, no. 1-2, pp. 19–21, r 90.

[13] J. Goldsmith, *Unsupervised Learning of the Morphology of a Natural Language*. USA: MIT Press, 2001.

[14] J. Mayfield and P. McNamee, "Single n-gram stemming," in *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*. New York, NY, USA: ACM Press, 2003, pp. 415–416.

[15] L. S. Larkey, M. E. Connell, and N. Abduljaleel, "Hindi CLIR in thirty days," *ACM Transactions on Asian Language Information Processing (TALIP)*, vol. 2, no. 2, pp. 130–142, 2003.

[16] W. Waller and D. H. Kraft, "A Mathematical Model of a Weighted Boolean Retrieval System," *Information Processing and Management*, 1979.

[17] K. Lund and C. Burgess, "Producing high-dimensional semantic spaces from lexical co-occurrence," in *Behavior Research Methods, Instrumentation, and Computers*, 1996, pp. 203–208.

[18] V. Lavrenko, M. Choquette, and W. B. Croft, "Cross-lingual relevance models," in *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM Press, 2002, pp. 175–182.