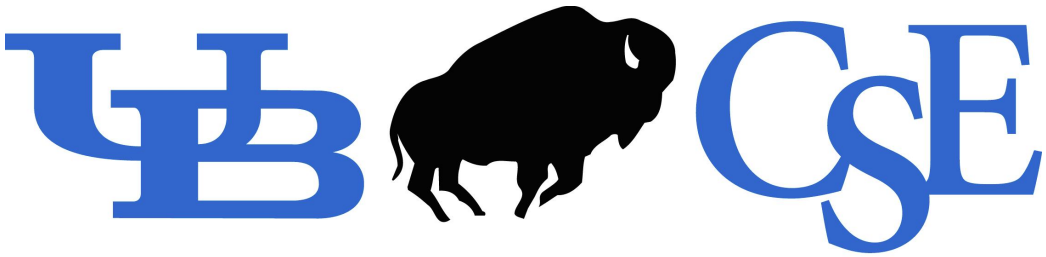
University at Buffalo

CSE 535 - Information Retrieval, Fall 2016

# Project 4: Implementation of Cross Lingual IR

**Team 63 Lisbon**
Anurag Devulapalli
Sandeep Shenoy
Vipin Kumar
Vivek Singh

December 8, 2016



Department of Computer Science and Engineering
University at Buffalo *The State University of New York*

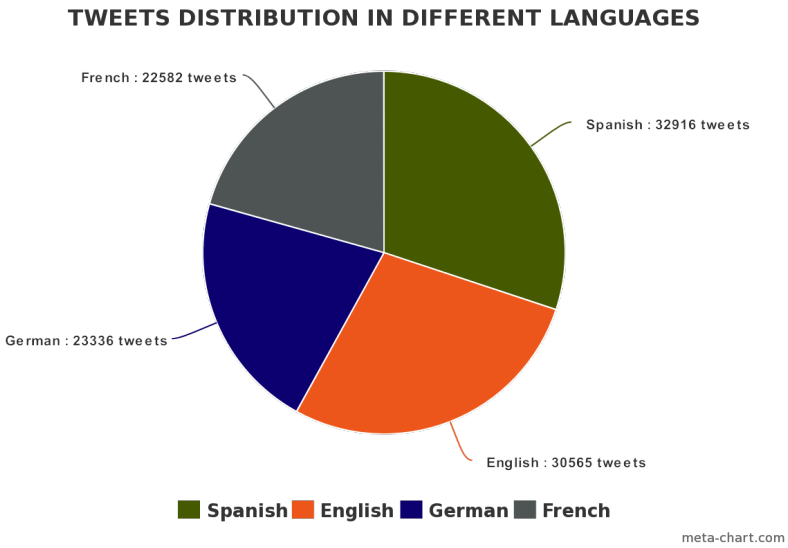
# Contents

# 1 Introduction

The project is an implementation of the research paper published by Fujii and Ishikawa (2000) for Cross Lingual content retrieval with Apache Solr 6.2.0 at its core.
The Cross Lingual search engine, allows users to query in any of the four different languages - English, Spanish, French, German and retrieve the data in any or all of the languages. The application searches for relevant content in a multilingual database of tweets and presents it to the user with reasonable precision. We have also worked to increase the browsing efficiency with the help of machine translation, result clustering and cross lingual auto complete feature.
In the following sections, we briefly describe the dataset used as Index, Preprocessing techniques, back end implementation and finally the front end features available to the end user.

# 2 Dataset

In contrast to the bilingual data set used by Fujii and Ishikawa (Japanese and English Patent data) , the data set in this project includes more than a hundred thousand Tweets collected over a span of 7 days in four different languages- English, Spanish, French and German. We have utilized the resources from the project 1 to retrieve the tweets using Twitter API.



# 3 Preprocessing

## 3.1 Detecting query and document language

Detecting query language was necessary to boost the results in native language of the query, since the user typing the query is more likely to be interested in the results in native language. Hence, we implement language detection at two stages. In the first stage we detect language of the documents during indexing using solr and in the second stage, we detect language of the query during query processing,using Langdetect API.

```
public String LangDetect(String text) throws IOException, JSONException{

        String charSet = java.nio.charset.StandardCharsets.UTF_8.name();
        String url="http://ws.detectlanguage.com/0.2/detect";
        String querytext=text;
        String LANG_DETECTION_API_KEY = "7bd4928c9e29cc17138d3c810f39c3a8";
        String JsonresponseLang="";
        String language = null;

        String query=String.format("q=%s&key=%s", URLEncoder.encode(querytext, charSet),
                URLEncoder.encode(LANG_DETECTION_API_KEY, charSet));
        JsonresponseLang=fetchHTTPData(url,query);

        if (JsonresponseLang==""){
            System.out.println("No response from Language detection server...");
        }else language = extractLanguage(JsonresponseLang);


        return language;
    }

    private String extractLanguage(String jsonresponseLang) throws JSONException {

        String language = "";
        JSONObject jsontext;


        jsontext = new JSONObject(jsonresponseLang);
        language=jsontext.getJSONObject("data").getJSONArray("detections").getJSONObject(0).
                get("language").toString();
        return language;
}
```

Figure 1: LangDetectLanguageIdentifierUpdateProcessorFactory

## 3.2 Deduplication

The initial corpora had a significant number of duplicate documents. To eliminate the duplicate results, we implemented deduplication strategy once in the preprocessing stage and again in post processing stage. In preprocessing, we filtered the retweeted documents before indexing it to solr. In post processing, we eliminated the rest of the duplicate documents using solr SignatureUpdateProcessorFactory. The signature was implemented us-

ing Lookup3Signature since it is faster than MD5 and works best on smaller index.

```
<processor class="solr.processor.SignatureUpdateProcessorFactory">
        <bool name="enabled">true</bool>
        <str name="signatureField">signature_field</str>
        <bool name="overwriteDupes">true</bool>
        <str name="fields">text,lang</str>
        <str name="signatureClass">solr.processor.Lookup3Signature</str>
  </processor>
```

Figure 2: Implementation of deduplication in solr

# 4 Back end implementation

## 4.1 Implementation model of Fujii and Isikawa

To apply monolingual retrieval techniques in our perspective, we chose to convert queries into document language, as with the model adopted by Fujii and Ishikawa.

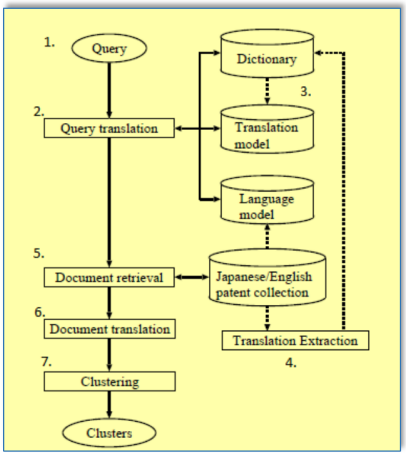



Figure 3: The figure describes the overall design of the model implemented by Fujii and Ishikawa in developing PRIME, which retrieves documents as per the user query,which could be in English and Japanese.

We now describe an implementation of a similar model to retrieve multilingual tweets as below:

### 4.1.1 Query

Query is received in any of the 4 languages, which is passed to solr without any preprocessing.

### 4.1.2 Query Translation

Translation of query is achieved by Google translate API. Out of all machine translation tools available at present, "Google Translate" proved to be most accurate in terms of grammatical structure of translation. We tried experimenting with "Microsoft Azure" API and Yandex API, but Google API proved to produce most relevant result, especially in case of German language, which contains long compound statements.

### 4.1.3 Translation model

This part of query translation is handled by Google translate API. We observed that the English to French/German translation were quite accurate in all the top three tools but Google translate provided the best result in reverse translation. Also since Google translate uses neural machine translation engine, which translates whole sentences at a time, rather than just piece by piece. It uses this broader context to figure out the most relevant translation, and then rearranges the words to ensure the grammatical correctness.This in fact is better than Nova dictionary as implemented in development of PRIME.

```
private String callUrlAndParseResult(String langFrom, String langTo,String word) throws Exception
        {

         String url = "https://translate.googleapis.com/translate_a/single?"+
           "client=gtx&"+
           "sl=" + langFrom +
           "&tl=" + langTo +
           "&dt=t&q=" + URLEncoder.encode(word, "UTF-8");

         URL obj = new URL(url);
         HttpURLConnection con = (HttpURLConnection) obj.openConnection();
         con.setRequestProperty("User-Agent", "Mozilla/5.0");

         BufferedReader in = new BufferedReader(
           new InputStreamReader(con.getInputStream()));
         String inputLine;
         StringBuffer response = new StringBuffer();

         while ((inputLine = in.readLine()) != null) {
          response.append(inputLine);
         }
         in.close();

         return parseResult(response.toString());
        }

        private String parseResult(String inputJson) throws Exception
        {

         JSONArray jsonArray = new JSONArray(inputJson);
         JSONArray jsonArray2 = (JSONArray) jsonArray.get(0);
         JSONArray jsonArray3 = (JSONArray) jsonArray2.get(0);

         return jsonArray3.get(0).toString();
        }
```

Figure 4: Implementation of Machine translation using Google API

### 4.1.4 Document retrieval

The retrieval model is based on Okapi BM25 which is implemented on the probabilistic retrieval framework developed by Robertson and Walker. We implemented word based indexing. Since the document length were short and almost fixed so we removed the document length normalization by fixing b value as 0 and increased the contribution of document frequency by fixing k value as 2.As per the test results, this combination of parameters provided the best result, effecting the precision in translation of English to German document retrieval.

### 4.1.5 Document Translation

Although the tweets were first retrieved in their original language, we provided an additional feature for document translation in our front end design. All the results could be translated and displayed in any other language.

### 4.1.6 Clustering

The model adopted by Fujii and Ishikawa implements Hierarchical Bayesian Clustering, however we implemented suffix tree clustering which is also an agglomerative hierarchical clustering algorithm based on information bottleneck method that tries to merge tweets based on mutual information (MI) between tweets clusters and terms. We decided to implement STC clustering because of the availability of carrot2 api and its integration in solr.

```
<searchComponent name="clustering" enable="true" class="solr.clustering.ClusteringComponent">

  <!-- An example definition for the STC clustering algorithm. -->
  <lst name="engine">
    <bool name="optional">true</bool>
    <str name="carrot.algorithm">org.carrot2.clustering.stc.STCClusteringAlgorithm</str>
    <str name="carrot.resourcesDir">clustering/carrot2</str>
    <str name="name">stc</str>
```

Figure 5: Implementation of STC clustering - search component

```
<requestHandler name="/clustering"
                startup="lazy"
                enable="${solr.clustering.enabled:false}"
                class="solr.SearchHandler">
  <lst name="defaults">
    <bool name="clustering">true</bool>
    <bool name="clustering.results">true</bool>
    <str name="clustering.engine">default</str>
    <!--<str  name="carrot.title">text</str> -->
    <!-- Logical field to physical field mapping. -->
    <str name="carrot.snippet">text</str>
    <bool name="carrot.produceSummary">true</bool>
    <str name="carrot.lang">lang</str>

    <str name="langid.map.lcmap">english:en french:fr spanish:es german:de</str>

    <!-- Configure any other request handler parameters. We will cluster the
         top 100 search results so bump up the 'rows' parameter. -->

    <str name="defType">edismax</str>
      <str name="qf">
        text^1.0
      </str>

    <str name="rows">100</str>
    <str name="fl">text,score</str>
  </lst>

  <!-- Append clustering at the end of the list of search components. -->
  <arr name="last-components">
    <str>clustering</str>
  </arr>
</requestHandler>
```

Figure 6: Implementation of STC clustering - request handler

## 4.2 Implementation in Solr

Implementation of the model is Solr required detecting language of the query and translate into the four languages. One of the way to implement is modifying the query after translating the query terms. This approach does not provide much flexibility. Hence, we wrapped our core implementation module in a custom solr plugin.

- We developed custom plugin for Edismax query parser. We chose Edismax query parser because of the below mentioned features:
  - It supports and/or queries, which is required for combining translated multi language query.
  - It supports improved boost functions.

  - It lets user specify which fields end user is allowed to query. This was required since we indexed complete twitter data in solr and having such feature allows limiting the user search fields.

- In order to support query result in native language of the query, we applied boost to the documents in native query language. We also boosted the queries based on the number of followers and hashtags.

```
public class ModifiedQueryParserGoogleApi2 extends ExtendedDismaxQParserPlugin  {

        static String ModifiedQuery = null;
        // Modified PF, PF1,Pf2 to boost native languages
        //adding user follower and user listed field to give boost to popular and verified accounts
        //boosting tweets with more followers and retweet
        static  String[] ModifiedPF = { "text_en^3", "text_es^3", "text_de^3", "text_ru^3", "followers^5","twee
        static  String[] ModifiedPF2 ={ "text_en", "text_es", "text_de", "text_ru", "followers^5","tweet_hashta
        static  String[] ModifiedPF3 ={ "text_en^2.5", "text_es^2.5", "text_de^2.5", "text_fr^2.5", "followers^

      //refer https://lucene.apache.org/solr/4_1_0/solr-core/org/apache/solr/search/ExtendedDismaxQParser.html
      //constructor details
      @Override
        public QParser createParser(String qstr,SolrParams localParams,SolrParams params,SolrQueryRequest req){


            ModifiedQuery=qstr;
            StringBuilder builder = new StringBuilder();
            builder.append(System.getProperty("line.separator"));

            String qstr_en  ="",qstr_es="",qstr_de="",qstr_fr="";
```

Figure 7: Implementation of custom solr plugin - 1

```
//Boosting native language of the query
if(lang.contains("en")){
    ModifiedPF[0]="text_en^6";
    ModifiedPF2[0]="text_en^2";
    ModifiedPF3[0]="text_en^4";
    try {
        qstr_en=callUrlAndParseResult("en", "en", qstr);
        qstr_es=callUrlAndParseResult("en", "es", qstr);
        qstr_de=callUrlAndParseResult("en", "de", qstr);
        qstr_fr=callUrlAndParseResult("en", "fr", qstr);
    } catch (Exception e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
}
if(lang.contains("es")){
    ModifiedPF[0]="text_es^6";
    ModifiedPF2[0]="text_es^2";
    ModifiedPF3[0]="text_es^4";
    try {
        qstr_en=callUrlAndParseResult("es", "en", qstr);
        qstr_es=callUrlAndParseResult("es", "es", qstr);
        qstr_de=callUrlAndParseResult("es", "de", qstr);
        qstr_fr=callUrlAndParseResult("es", "fr", qstr);
    } catch (Exception e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
}
if(lang.contains("de")){
    ModifiedPF[0]="text_de^6";
    ModifiedPF2[0]="text_de^2";
    ModifiedPF3[0]="text_de^4";
    try {
        qstr_en=callUrlAndParseResult("de", "en", qstr);
        qstr_es=callUrlAndParseResult("de", "es", qstr);
        qstr_de=callUrlAndParseResult("de", "de", qstr);
        qstr_fr=callUrlAndParseResult("de", "fr", qstr);
    } catch (Exception e1) {
```

Figure 8: Implementation of custom solr plugin - 2

# 5 Front end features

## 5.1 Faceting

Faceting makes it easier for users to explore search results, narrowing in on exactly the results they are looking for. To arrange the search results into categories for ease of access to searchers, we have implemented faceting in our website. Faceting was handled entirely in the front end by categorizing the search results through a java program.


Figure 9: Search for documents by language(s) or region(s)

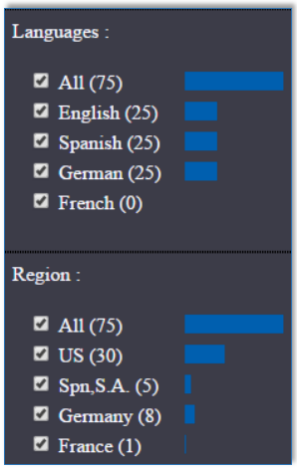



Figure 10: Filter retrieved results by language(s) or region(s)

## 5.2 Cross Lingual Query Suggestion

To further improve user experience and enhance the precision of search results, we enhanced our cross lingual custom plugin to develop a multilingual autocomplete feature from scratch. The unique feature of the suggestion box being that the search result is interleaved in query suggestion. The suggestion shown are in fact indexed twitter data retrieved in real time.

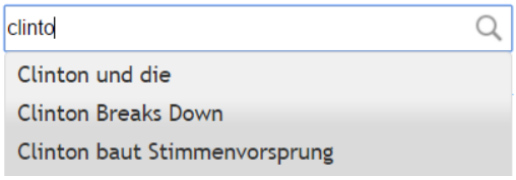



Figure 11: Autocomplete feature

## 5.3 Search Result Translation

As the retrieved results could be multilingual, we have provided a functionality to translate all of the retrieved results or a single result to the desired language.

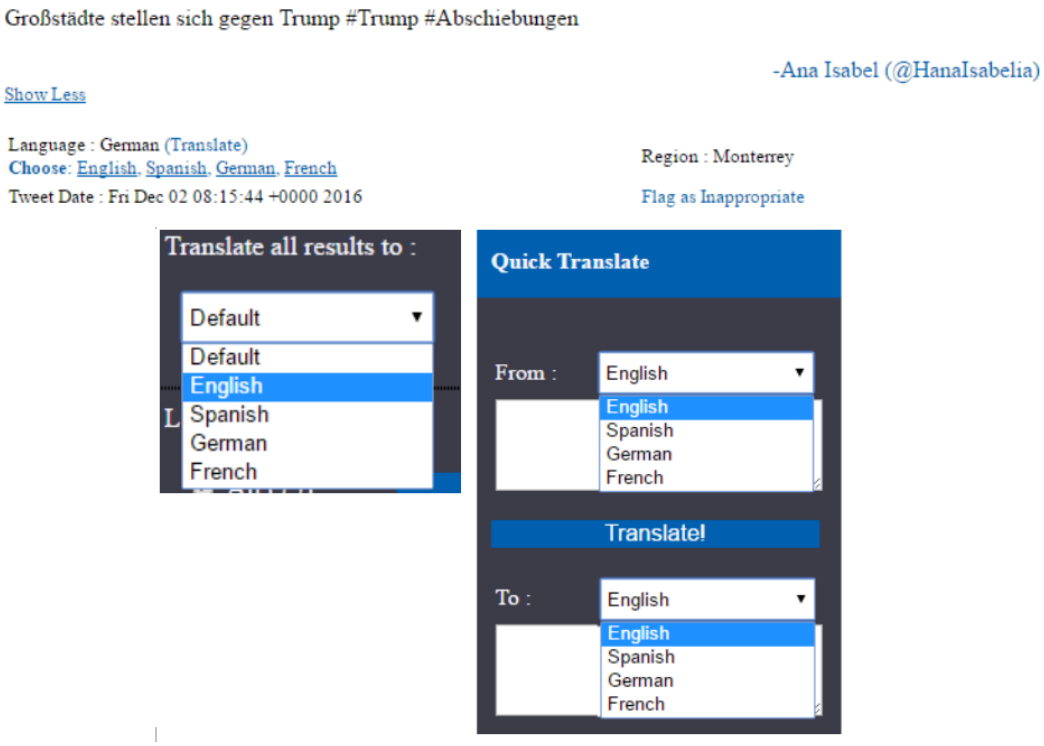



Figure 12: Result translation at various levels, allowing the user to translate a single tweet, phrase or all the retrieved results to the language selected by the user.

# 6 Member Contributions

- Sandeep D Shenoy (50205705)
  - Complete User Interface design and Implementation of Chirpy Bird.
  - Implementing Faceting logic in Front End to categorize tweets.
  - Video Editing.
  - Presentation.
- Vipin Kumar (50208397)
  - Implementing the connection between front end and Solr.
  - Receiving and formatting retrieved Json tweets.
  - Developed language translation for front End.
  - Developed Autocomplete feature in UI.
- Vivek Singh (50208473)
  - Understanding Implementing CLIR model in Solr.
  - Developing custom plugin for query modification.
  - Implementing language detection and translation in backend.
  - Preparing Project report.
- Anurag Devulapalli (50208153)
  - Collecting tweets.
  - Implementing Deduplication in Solr.
  - Attempted result clustering using Carrot 2.
  - Code deployment and handling in Server.
  - Formatting and finalizing report.

# References

[1] S. Higuchi, M. Fukui, A. Fujii, and T. Ishikawa. PRIME: A system for multi-lingual patent retrieval. In *Proceedings of MT Summit VIII*, pages 163-167, 2001.