Automating high-quality translations for Mobile Apps

Thesis subtitle

Master Thesis

Wanzenried & Stefan
from
Bern BE, Switzerland

Philosophisch-naturwissenschaftliche Fakultät
der Universität Bern

Month and Year

Prof. Dr. Philippe Cudr-Mauroux, eXascale Infolab, University of Fribourg, Switzerland, Supervisor
Roman Prokofyev, eXascale Infolab, University of Fribourg, Switzerland, Assistant

Abstract

Every day, hundreds of mobile applications are added to stores such as Google Play or AppStore. Many of them are intended to be used internationally, and thus require translation of the interface. At the same time, many more mobile apps already available for download in these stores. Leveraging the translation bases of existing applications, we could immediately provide high-quality translations for new apps without need to go though a human-translation process.

This project aims to extract and parse translations of existing applications to see if they can be used to translate new ones. A prototype application allows to translate existing apps from a given source to a target language. Translations are generated with different algorithms in statistical machine translation.

Contents

1 Introduction

Android, iPhone and Windows smart phones are available in countries all over the world. If an app is intended to be used internationally, translating it into multiple languages can be important to attract users. Users may only use the app if its text is available in their native language. Offering multiple languages is therefore crucial for paid apps to increase sales. However, translating an app requires additional effort. From a developers perspective, all strings must be isolated in order to replace them with different values for each supported language. Furthermore, translating text itself is a difficult task which usually requires the work of a professional translator.

The main idea of this thesis is to collect a lot of existing apps and extract their translations. Are we able to provide high quality translations with the help of Statistical Machine Translation (SMT)? To reduce the scope of different mobile operating systems and languages, we focus on automatic translations of Android apps from English to French and German. From now on, the term *app* in this thesis refers to a Android application. We have chosen French and German as target languages for the translations because they are natively spoken in Switzerland, which makes understanding the produced results easier. SMT is the dominant approach in automating translations and used by services like "Google Translate"[1] and Microsoft's "Bing Translator"[2]. In this thesis, we use different SMT algorithms to produce translations and compare their results, both manually and automatically. A prototype web application is built which serves as a playground to translate strings and apps.

[1] `https://translate.google.com`
[2] `https://www.bing.com/translator`

2 Problem Analysis

This chapter answers the question why it is important to offer multiple languages for apps.

2.1 Importance of Translating Mobile Apps

Google encourages developers to localize their apps on their localization checklist.[1]

> *Android and Google Play offer you a worldwide audience for your apps, with an addressable user base that's growing very rapidly in countries such as Japan, Korea, India, Brazil, and Russia. We strongly encourage you to localize as it can maximize your apps distribution potential resulting in ratings from users around the world.*

The developer should first identify the countries where the app will be distributed based on the overall market size and opportunity, app category, local pricing etc. According to the identified countries, the supported languages are determined. Ideally an app supports multiple languages already before the first version is available in the app store. Users may install the app only once and forget about it, if they are not able to understand it.

Since 2013, Google offers an (human-based) app translation service to developers.[2] Several developers who participated at the pilot program shared their results after translating their apps:

- The developers of Zombie Ragdoll[3] used this tool to launch their new game simultaneously in 20 languages in August 2013. When they combined app translation with local marketing campaigns, they found that 80% of their installs came from non-English-language users.

[1] Android localization checklist: `http://developer.android.com/distribute/tools/localization-checklist.html`

[2] Android Developers Blog post about Googles translation service: `http://android-developers.blogspot.ch/2013/11/app-translation-service-now-available.html`

[3] Game Zombie Ragdoll in the Google Playstore: `https://play.google.com/store/apps/details?id=com.rvappstudios.zombieragdoll`

- Dating app SayHi[4] Chat expanded into 13 additional languages using the App Translation Service. They saw 120% install growth in localized markets and improved user reviews of the professionally translated UI.
- The developer of card game G4A Indian Rummy[5] found that the App Translation Service was easier to use than their previous translation methods, and saw a 300% increase with user engagement in localized apps.

In 2012, "App Annie" [6] (former called "Distimo"), published a study where they analysed the impact of translations to iPhone apps.[7] They found out that the use of native language is still limited in several countries, for example Italy, Russia or Brazil. Native-only apps are mostly used in Asian countries (see Figure 2.1). The study followed 200 iPhone apps that introduced native language support in a market and observed their growth. Only one week after the local language support was introduced with an app update, downloading increased by 128% and revenue by 26%.

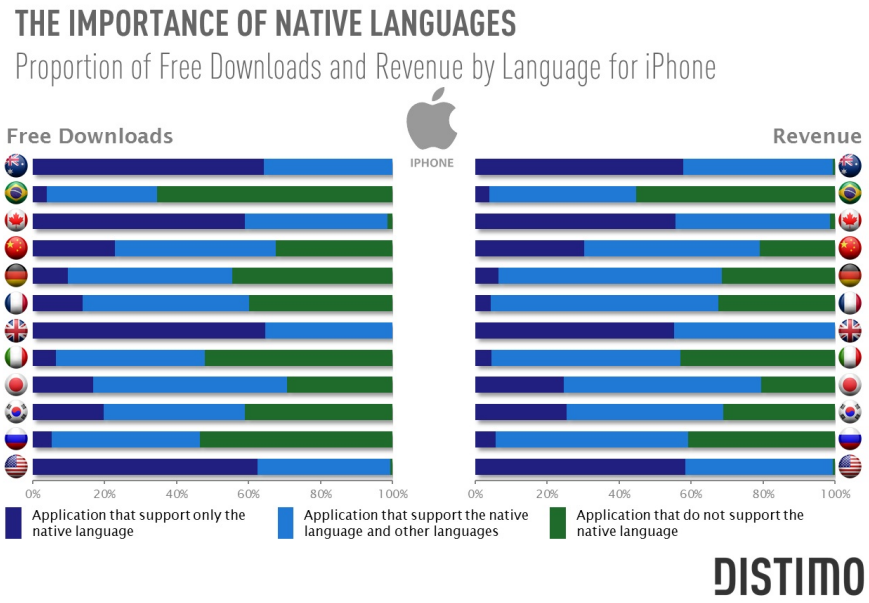


Figure 2.1: Screenshot from the study "The impact of app translations" by App Annie showing the distribution of apps supporting native and other languages for the 12 top languages.

[4]Dating app SayHi! in the Google Playstore: `https://play.google.com/store/apps/details?id=com.unearby.sayhi`

[5]Card game Indian Rummy `https://play.google.com/store/apps/details?id=org.games4all.android.games.indianrummy.prod`

[6]`https://www.appannie.com`

[7]App Annie's study The impact of app translations: `https://web.archive.org/web/20150327084808/http://www.distimo.com/publications`

2.2 Translation Services

2.3 Statistical Machine Translation

Machine translation has a long history, but over the last decade or two, its evolution has taken on a new direction - a direction that is mirrored in other subfields of natural language processing. This new direction is grounded in the premise that language is so rich and complex that it could never be fully analyzed and distilled into a set of rules, which are then encoded into a computer program. Instead, the new direction is to develop a machine that discovers the rules of translation automatically from a large corpus of translated text, by pairing the input and output of the translation process, and learning from the statistics over the data.[2]

SMT translation models are usually trained on a large parallel corpus which is freely available on the web [8]. The BLEU metric is then used to measure the quality of the produced translations. [5] There exists a recurring translation task of the "Workshop on statistical machine translation", which focuses on translating european language pairs by improving existing systems[9].

In this thesis, we compare two different SMT algorithm and evaluate how they perform in the context of translating short sentences of mobile apps. The models are therefore applied on a much smaller data set than usual.

2.4 Challenges

- Quality of existing translations
- Where and how to obtain large amount of apps
- Hardware to train the translation models
- How many data is needed to produce good results?
- How can we measure good translations?

[8] http://www.statmt.org/europarl/
[9] http://www.statmt.org/wmt15/translation-task.html

3
Related Work

This chapter provides an overview of Statistical Machine Translation (SMT). We consider two different models. The first one is called "Statistical Phrase-Based Translation" and was introduced in 2003 by Philipp Koehn et al. [1][2] The second, more recent model, uses Deep Learning Neural Networks to perform translations. [6]

3.1 Phrase-Based Machine Translation

3.1.1 Model

Given the task to translate a sentence from a foreign language $f = f_1, f_2, f_3..., f_m$ into an english sentence $e = e_1, e_2, e_3..., e_n$, the translation model is based on the following probabilistic equation:

$$P(e|f)$$

To find the best english translation, the following equation is used (Bayes rule).

$$\begin{aligned} e_{best} &= argmax_e P(e|f) \\ &= argmax_e \frac{P(f|e)P(e)}{P(f)} \\ &= argmax_e P(e|f)P(e) \end{aligned}$$

where $P(e)$ is called the *language model*[1] and $P(f|e)$ is the *translation model*.
The translation model is further decomposed:

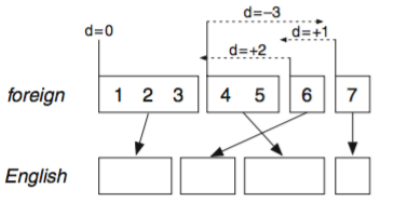
$$p(\bar{f}_1^I|\bar{e}_1^I) = \prod_{i=1}^{I} \phi(\bar{f}_i^I|\bar{e}_i^I) d(start_i - end_{i-1} - 1)$$

where ϕ is called the *phrase translation probability* and d is the *reordering probability*.

[1] https://en.wikipedia.org/wiki/Language_model

The foreign sentence *f* is broken up into *I* phrases $\bar{f}_i$. Each foreign phrase is translated into an English phrase $\bar{e}_i$.

Reordering is handled by a *distance based reordering model*. We define $start_i$ as the position of the first word of the foreign input phrase that translates to the *i*th English phrase, and end_i as the position of the last word of that foreign phrase. The reordering distance is the number of words skipped when taking foreign words out of sequence (see figure 3.1).



Phrase	Translates	Movement	Distance
1	1–3	start at beginning	0
2	6	skip over 4–5	+2
3	4–5	move back over 4–6	−3
4	7	skip over 6	+1

Figure 3.1: Distance based reordering [2]

d is turned into a proper probability distribution by applying an exponentially decaying cost function $d(x) = \alpha^{|x|}$ with $\alpha\epsilon[0, 1]$. This means that movements of phrases over large distances are more expensive than shorter movements or no movement at all.

3.1.2 Learning a Phrase Translation Table

Building a phrase translation table involves three stages:

1. **Word alignment** The words of each sentence from the parallel corpus of the foreign and English language must be aligned in order to extract phrase pairs. The alignment is typically done by using one of the IBM models.[2] Figure 3.2 shows an example of word alignment for a top term from our corpus.

2. **Extraciton of phrase pairs** Extract all phrase pairs of parallel sentences that are consistent with word alignment. A phrase pair ($\bar{f}$,$\bar{e}$) is consistent with an alignment *A*, if all words $f_1, ..., f_n$ in $\bar{f}$ that have alignment points in *A* have these with words $e_1, ..., e_n$ in $\bar{e}$ and vice versa.

3. **Scoring phrase pairs** Assign probabilities to phrase translations.

[2]IBM word alignment models: `https://en.wikipedia.org/wiki/IBM_alignment_models`

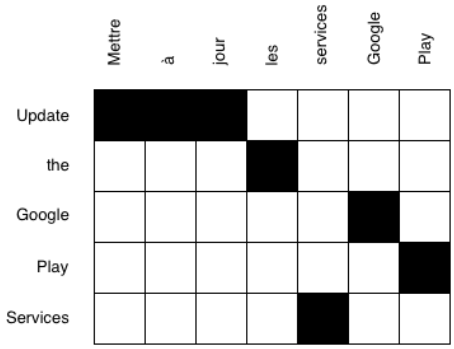


Figure 3.2: Word alignment between a top term of our parallel Enlgish/French corpus.

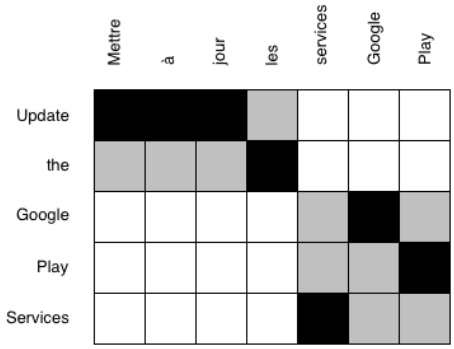


Figure 3.3: Extract phrase pairs consistent with word alignment.

We are able to extract the following phrase pairs from the example sentence:

- Update the Google Play Services / Mettre à jour les services Google Play
- Update / Mettre à jour
- Update the / Mettre à jour les
- Google Play Services / services Google Play
- The Google Play Services / les services Google Play
- Google Play / Google Play
- Google / Google
- Play / Play
- Services / services
- the / les

The last step is to estimate the phrase translation probabilities by the relative frequency:

$$\phi(\bar{f}|\bar{e}) = \frac{count(\bar{e}, \bar{f})}{\sum_{\bar{f}_i} count(\bar{e}, \bar{f}_i)}$$

These probabilities are put in the phrase translation table, together with the English and foreign phrase (see Figure 3.1).

Foreign Phrase	$\phi(\bar{f}\|\bar{e})$
paramètres	0.9
réglages	0.36
configuration	0.15

Table 3.1: Example of a phrase translation table containing the estimated probabilities for translating the English word "Setting" to French.

3.1.3 Decoding

Decoding is the process of finding a translation for a given input sentence. Recall the model to compute the translation probability:

$$e_{best} = argmax_e \prod_{i=1}^{I} \phi(\bar{f}_i|\bar{e}_i) d(start_i - end_{i-1} - 1) p_{LM}(e)$$

- **Phrase translation** Pick phrase $\bar{f}_i$ to be translated as phrase $\bar{e}_i$ by looking up the score from the phrase translation table.
- **Reordering** The previous phrase ended in end_{i-1}, the current phrase starts at $start_i$. Compute $d(start_i - end_{i-1} - 1)$
- **Language model** For a *n*-gram model, we need to keep track of the last $n-1$ words. The score is computed for every added word w_i: $p_{LM}(w_i|w_i - (n-1), ..., w_{i-1})$

The algorithm reads the input sentence from left to right and creates a new hypothesis by picking any phrase translation at a time. The score is computed incrementally for each partial hypothesis. This is repeated recursively until all hypotheses have been expanded. A hypothesis covering all input words cannot be expanded further and forms and end point in the search graph (Figure 3.4). To get the best translation, we pick the completed hypothesis with the highest probability score and backtrack through the search graph.

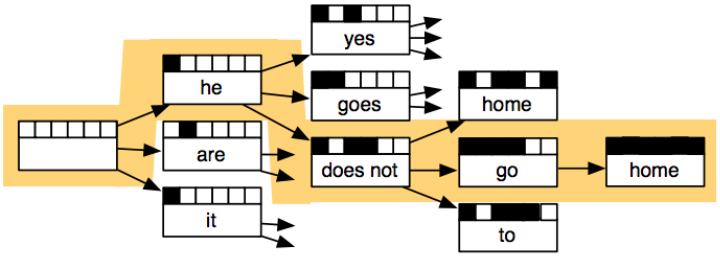


Figure 3.4: Example: Created hypothesis by translating the German sentence “er geht ja nicht nach hause” to English. The squares on top indicate a coverage vector of translated German words (filled black if covered). [2]

This process creates an exponential number of hypothesis. To reduce the search space, the following methods are applied:

- **Recombination** Two hypothesis paths lead to two matching hypotheses, e.g. have the same English words in the output. The worse hypothesis is dropped.
- **Stack Pruning** Bad hypotheses are removed early. Hypothesis that have translated the same number of input words are put into a stack, so that they are comparable. The number of hypotheses in a stack is limited, for example by keeping only k hypotheses according to a score. This score is a combination of the partial probability score and a future cost estimation of the remaining sentence.

3.2 Deep Learning Neural Networks

4 Data Analysis

4.1 Apps

A total of 698 free Android apps were collected from various categories[1], ensuring a good mix of different translations. Most of them were chosen randomly beside the top apps, mostly from Google. Figure 4.1 shows the top ten apps according to the number of available english translations. Note that the counts indicate the number of effective translations after eliminating uninteresting data during pre-processing the corpus (see chapter 5). Most of the apps have similar counts for all the three languages. Surprisingly, some apps provide more translations for german and french than english. Different counts can happen because of:

1. Not all strings were translated to all languages. Some english words remain the same in the target language and don't need to be translated, e.g. E-Mail.
2. The XML files containing the translations are out of sync, e.g. the developer forgot to add a french translation.
3. Some translations were abandoned during pre-processing.

34 apps did not isolate their strings in XML files (we assume that they are hardcoded in the programming code). This leaves a effective number of 664 apps where we could extract translations. However, 241 apps among them were not multilingual and therefore not useful for building translation systems. Unfortunately, apps are not marked as multi-language in the app store, so we only know if there are translations available after downloading and extracting the data.

[1] Click on Categories to see a list of available categories at: `https://play.google.com/store/apps?hl=en`

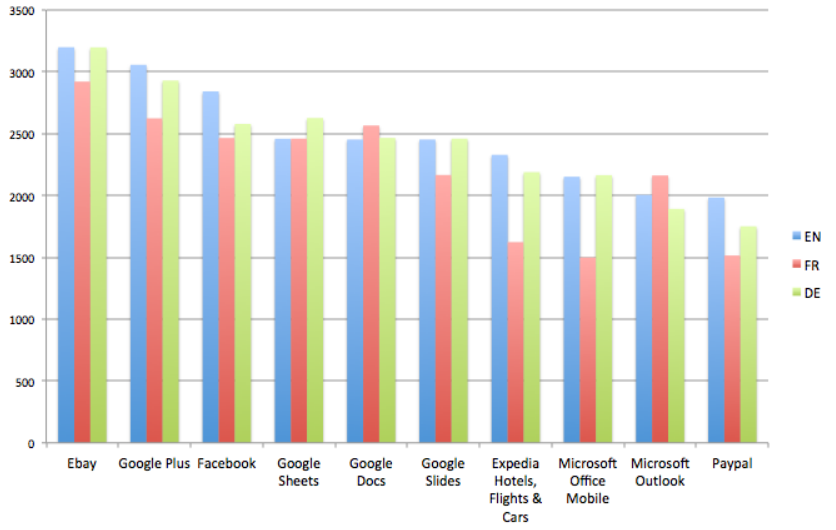


Figure 4.1: Top apps by the number of english translations

4.2 Languages

We collected a total of 423 multilingual apps, offering its content at least for two languages. Figure 4.2 shows the number of apps for the ten most used languages. French and German, where we focus on this thesis, are both in the top 4. The top apps are translated in many more languages. Overall, 117 different languages were available.

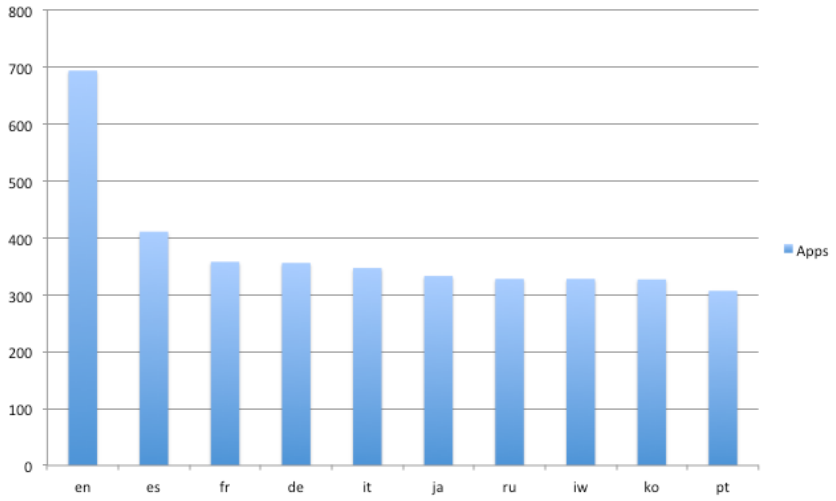


Figure 4.2: Number of apps available in the top ten languages.

4.3 Top Translations

EN	Count	FR	Count	DE	Count
Cancel	1894	Annuler	1067	Abbrechen	886
Done	1230	Supprimer	745	Google Play-Dienste aktivieren	730
Settings	1102	Connexion	745	Google Play-Dienste installieren	728
Search	884	Mettre jour	486	Anmelden	664
Delete	704	Rechercher	485	Lschen	664
Enable Google Play services	692	Paramtres	450	Aktualisieren	652
Get Google Play services	690	Mettre jour les services Google Play	370	Einstellungen	569
Log Out	556	Activer services google Play	368	Fertig	548
Update	545	Activer les services Google Play	366	Weiter	484
Share	541	Installer les services Google Play	366	Schliessen	428

Table 4.1: Top ten translations for english, french and german

5 Data Extraction, Preparation and Storage

5.1 Extract Translations

An Android app has the form of a single binary file (suffix .apk). The XML files containing the translations are packed inside the .apk file. We used a software called "Apktool"[1] to reverse engineer existing apps and extract the needed XML files.

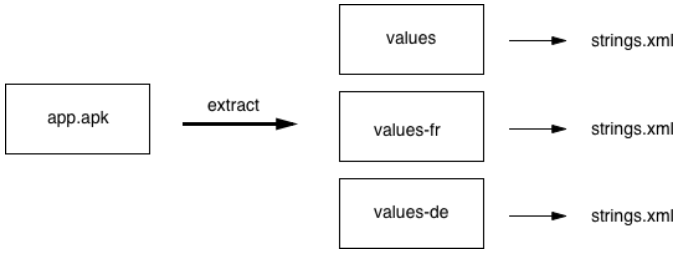


Figure 5.1: For each language, the translations are stored in a XML file

Inside the `strings.xml` files, translations are stored key-value based. Each string has a unique key which holds the translation value for each language. Here is an example how an English XML file looks like:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="close_app">Close Application</string>
    <string name="back">Back</string>
    ...
</resources>
```

[1] `http://ibotpeaches.github.io/Apktool/`

5.2 Preprocess Translations

This task involves two parts. First of all, the translations are sanitized from unwanted information such as HTML tags. Some strings are omitted at this stage because they do not provide any meaningful value to the translation process, for example URLs or string placeholders. Secondly, the sanitized strings are tokenized so that they can be used by an SMT system.

5.2.1 Sanitizing

The following rules are used to get rid of unwanted translations or parts of it:

- Trim the string from newlines (`\n`), tabs (\t) or carriage return (\r)
- Strip any HTML tags
- Omit strings that are URLs (starting with http or www)
- Remove strings placeholders like `%s` or `%1$s`
- Ignore words with less than 3 characters
- Finally, the resulting string is trimmed again from spaces and must contain at least one alphanumerical character

5.2.2 Tokenization and Truecasing

Tokenization is required to separate words from punctuation. Each token is separated by a space character: "`Hi, my name is John.`" becomes "`Hi , my name is John .`"

After tokenizsation, truecasing was applied to our text corpus.[2] In contrast to lowercasing, truecasing determines the proper capitalization of words. This is done by first analyzing all data and then apply the most likely capitalization. It is especially useful for words starting a sentence, that are written uppercase in many languages. For example the word "`This`" starting a sentence will become "`this`", because it is usually written lowercase.

5.3 Data storage

5.3.1 Parallel and Monolingual Corpus

After pre-processing the translations, they are written to text files, one sentence per line. The corpus is split into parallel and monolingual data. A parallel corpus (often also called bitext) describes sentence aligned text-files of one language pair. All sentences must be aligned so that line x of the translation source language corresponds to line x of the target language. The sentences are shuffled before writing to get a random mix of translations of different apps. Most SMT system use this parallel data to train their translation model, so the parallel corpus can be reused multiple times.

In addition, data of any language is also stored monolingual. All sentences of one language are written into a single text file. This is for example used to build language models, which are an important part of a phrase-based translation model (see chapter 3.1).

[2] `https://en.wikipedia.org/wiki/Truecasing`

5.3.2 Apache Solr

We needed a simple way to store all translations, preferably with additional meta data, to perform data analysis and calculate statistics. Apache Solr[3] is used for this purpose. Solr is an open source text search platform built on top of Apache Lucene[4]. It allows to store the translations with additional meta data, such as the translation key and a unique app ID. Due to its way of indexing text data, Solr is very fast when querying for translation strings. Furthermore, built in tools allow to get nice statistics, for example the top terms grouped by language.

Solr is also used as Baseline translation system, where we translate strings with a simple algorithm directly from the collected data in Solr. (see chapter xy).

5.3.2.1 Setup and Schema

Solr offers so called "Cores" where each core manages a separate index, schema and configuration[5]. In our setup we created one core per language and one document per translation string. The schema is identical for each core (language) and consist of the following fields:

- **id** Unique ID used by Solr to identify a document.
- **app_id** Each app must have a unique app-ID.
- **key** The translation key from the XML file.
- **value** The translation value corresponding to the above key.
- **value_lc** Again the translation value, additionally indexed with a special start- and ending delimiter. This allows to search for exact values.

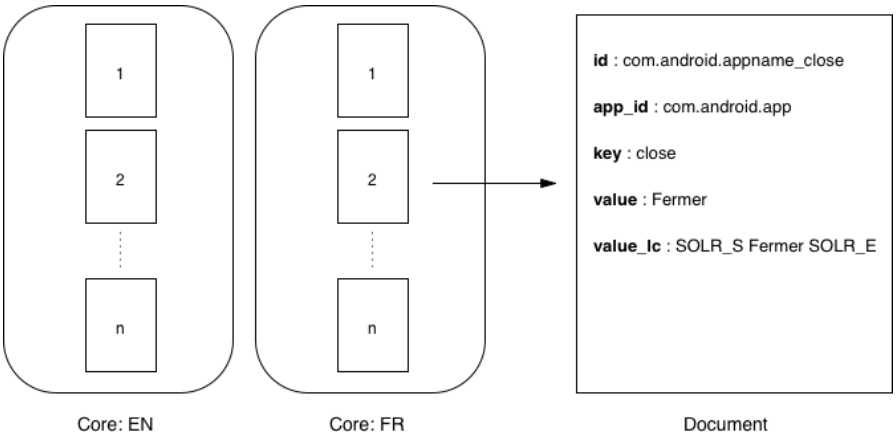


Figure 5.2: Translations are stored in a document inside a separate core per language

The data in the `value` and `value_lc` fields is stored tokenized and lowercased. The Solr Standard Tokenizer[6] splits text into tokens, treating whitespace and punctuation as delimiters. Delimiter characters are discarded.

[3] `http://lucene.apache.org/solr/`
[4] `https://lucene.apache.org/core/`
[5] `https://cwiki.apache.org/confluence/display/solr/Solr+Cores+and+solr.xml`
[6] `https://cwiki.apache.org/confluence/display/solr/Tokenizers#Tokenizers-StandardTokenizer`

5.3.2.2 Queries

Solr offers a REST based interface to query the database. There is also a built in web application which allows to inspect results. Here is an example query where we search for the string “Log Out” in the English core[7]. The maximum amount of results is set to 500 rows and the return type is JSON:

```
http://diufpc114.unifr.ch:8983/solr/en/select?q=value:"Log%20Out"&rows=
500&wt=json&indent=true
```

This query returns a total of 458 results. Note that there are not only exact matches returned, the term “Log Out” can occur as substring in a longer sentence (see figure xy). By default, Solr does not offer the possibility to return only exact matches. However, we can simulate this behaviour with the help of our `value_lc` field, which indexed the string together with a special start- and end delimiter (`SOLR_S` and `SOLR_E`):

```
http://diufpc114.unifr.ch:8983/solr/en/select?q=value_lc:"SOLR_S%20LogOut%
20SOLR_E"&rows=500&wt=json&indent=true
```

This query returns only 396 results. Note that the case of the search term doesn’t matter and whitespace and punctuation are ignored. Executing the above query with the term “log Out .” returns the same results.


```
"response": {
    "numFound": 458,
    "start": 0,
    "docs": [
      {
        "id": "com.flaregames.gargoyles_com_facebook_loginview_log_out_action",
        "app_id": "com.flaregames.gargoyles",
        "key": "com_facebook_loginview_log_out_action",
        "value": "Log Out",
        "value_lc": "SOLR_S Log Out SOLR_E"
      },
      {
        "id": "com.instagram.android_must_log_out_one_click_login",
        "app_id": "com.instagram.android",
        "key": "must_log_out_one_click_login",
        "value": "You must log out in order to login using this link.",
        "value_lc": "SOLR_S You must log out in order to login using this link. SOLR_E"
      },      ...
      ]
}
```


[7] You must be connected to the LAN of the University of Fribourg in order to access the referenced domain. Also note that the parameters should be fully url-encoded, this was omitted in this report for better readability.

6 Translation Process

This chapter describes the different translation systems that were used to translate apps.

6.1 Baseline System

The "Baseline System" makes direct use of the data stored in Solr and produces translations with a simple algorithm. Remember how the data is organized in Solr (see section 5.3.2): Each translation is stored in a document with a fixed schema and each document belongs to one core. To lookup a translation, the following procedure is used:

1. Search the string to translate in the source language
2. Parse the result documents and extract the `app_id` and `key` fields
3. Lookup translations in the target language by querying the target core with the collected `app_ids` and `keys`.
4. Count the number of unique translations found. The best translation is the value that was mostly used.

6.1.1 Algorithm

Given a string $S = [\ s_1\ s_2\ ...\ s_n\ ]$ with words s_i in the source language, we want to produce a string $T = [\ t_1\ t_2\ ...\ t_n\ ]$ in the target language. The algorithm recursively tries to translate the longest substrings of S by looking up translations for the substrings in Solr. This process is repeated until there are left only single words s_i to translate.

1. Check if we find any translation for S directly. If so, return the translation and exit.
2. Start translating the longest substrings of S. To build the substrings, we move a "window" of size $length(S) - i$ from left to right over S, where i is the iteration level. Let $length(S) = 7$; the following two substrings of length 6 are possible at the first iteration ($i = 1$):

$$V_1 = [\ s_1\ s_2\ s_3\ s_4\ s_5\ s_6\ ]\ [\ s_7\ ]$$
$$V_2 = [\ s_1\ ]\ [\ s_2\ s_3\ s_4\ s_5\ s_6\ s_7\ ]$$

Now, we search for translations for all substrings in both variations V_1 and V_2. If there are no translations available, the substring (window) size is reduced by one for the second iteration ($i = 2$):

$$V_1 = [\ s_1\ s_2\ s_3\ s_4\ s_5\ ]\ [\ s_6\ s_7\ ]$$
$$V_2 = [\ s_1\ ]\ [\ s_2\ s_3\ s_4\ s_5\ s_6\ ]\ [\ s_7\ ]$$
$$V_3 = [\ s_1\ s_2\ ]\ [\ s_3\ s_4\ s_5\ s_6\ s_7\ ]$$

This process of reducing the substring length is repeated until we can translate a substring of at least one variation V_j.

3. Let's consider what happens if there are translations available. For each variation V_j, we count the number of available translations for their substrings $\bar{S}_i$. We continue with the variation $\tilde{V}$ having the highest total count of translations among their substrings:

$$\tilde{V} = argmax \sum_i count\_translations(\bar{S}_i)$$

If there is a translation available for each substring $\bar{S}_i$ of $\tilde{V}$, we can build the output string T directly by concatenating the substring translations:

$$T = \bar{T}_1\ \bar{T}_2\ \bar{T}_3\ ...\ \bar{T}_n$$

Otherwise, we build the output string T incremental: If there is no translation found for substring $\bar{S}_i$, we set $S = \bar{S}_i$ and recursively call the algorithm again. If there are left single words to translate and no translation exists, we insert the source word into T by setting $t_i = s_i$.

This simple algorithm works well for translating short words or sentences where a direct translation exists in the target language. On the other hand, it suffers from the following problems if translating substrings is necessary:

- The sentences or words for the translated substrings are inserted at the same position in T where they occurred in the source string. This won't produce good results since words often need to be reordered depending on the target language, e.g. in German // TODO Example
- Bad translations can have a direct impact on the result. For example, due to an error or uncompleted translation in the XML files, the English word "the" could be mapped to a sentence with 20 words.

6.1.2 Example

Let us translate the source string "Open the settings to change your username" from English to French.

1. Check for a direct translation: No French translation available.
2. Start translating the substrings. The upper index indicates the number of translations found:

V_1 = [Open the settings to change your] ^{0} [username] ^{28}
V_2 = [Open] ^{63} [the settings to change your username] ^{0}

Continue with Variation 2 since the count (63 + 0) is higher than the count from Variation 1 (0 + 28).

3. We set T = [ouvrir] and S = [the settings to change your username] and go recursive. Again, there does not exist a translation for S directly so the substrings are built:

$$V_1 = [\text{ the settings to change your }]^{0}\ [\text{ username }]^{28}$$
$$V_2 = [\text{ the }]^{1}\ [\text{ settings to change your username }]^{0}$$

4. T = [ouvrir ... nom d'utilisateur] ; S = [the settings to change your]

$$V_1 = [\text{ the settings to change }]^{0}\ [\text{ your }]^{2}$$
$$V_2 = [\text{ the }]^{1}\ [\text{ settings to change your }]^{0}$$

5. T = [ouvrir ... votre nom d'utilisateur] ; S = [the settings to change]

$$V_1 = [\text{ the settings to }]^{0}\ [\text{ change }]^{26}$$
$$V_2 = [\text{ the }]^{1}\ [\text{ settings to change }]^{0}$$

6. T = [ouvrir ... modifier votre nom d'utilisateur] ; S = [the settings to]

$$V_1 = [\text{ the settings }]^{5}\ [\text{ to }]^{2}$$
$$V_2 = [\text{ the }]^{1}\ [\text{ settings to }]^{0}$$

7. T = [ouvrir les paramètres vers modifier votre nom d'utilisateur] ; S = [ ]

At the last step, there are translations available for both substrings of V_1 and the algorithm derived the following string: "ouvrir les paramètres vers modifier votre nom d'utilisateur"

6.2 Moses

Moses [3] [1] is a open source SMT toolkit. The training process follows closely the phrase-based model described in section 3.1. The source code is available on GitHub [2]. This project is very active, there are frequent commits to the master branch by 72 contributors and documentation on their website is excellent.

6.2.1 Training

The training process requires sentence aligned parallel data to build the translation models, and monolingual data for the language models. We can make use of the sanitized and tokenized data as described in section 5.2. The training process involves the following steps.

1. Build a language model from the monolingual data of the target language. Moses includes the KenLM toolkit [3] for this task.

[1] http://www.statmt.org/moses/
[2] https://github.com/moses-smt/mosesdecoder
[3] https://kheafield.com/code/kenlm/

2. Align words of the parallel sentences. We used a external library MGIZA [4], an extension of the popular GIZA++ word alignment toolkit [4]. MGIZA is multi-threaded and runs faster on a multi-core machine.

3. Extract phrases, all phrases are dumped into one big file. The following listing shows some example content of this file. Each line contains: English phrase, French phrase, Alignment points of matching words (English-French).

```
as you can see from |||  comme vous pouvez le voir ||| 0-0 1-1 2-2 3-3 4-5
as you can see |||  comme vous pouvez le voir ||| 0-0 1-1 2-2 3-3 4-5
as you can |||  comme vous pouvez le ||| 0-0 1-1 2-2 3-3
as you can |||  comme vous pouvez ||| 0-0 1-1 2-2 3-3
as you |||  comme vous ||| 0-0 1-1 2-2
as |||  comme ||| 0-0 1-1
```

4. Score phrases by computing the phrase translation probability $\phi(\bar{f}|\bar{e})$ and build the phrase table. Here is an example how the content of the phrase table is organized in Moses (Translating from English to French):

```
your version of Google Play |||  votre version de Google Play ||| 1 0.04 1 0.22
your version of Google |||  votre version de Google ||| 1 0.04 1 0.24
your version of |||  votre version de ||| 1 0.0427528 1 0.245609
your version |||  votre version ||| 1 0.635219 1 0.508584
your |||  ton ||| 1 0.591003 0.0434783 0.00469507
your |||  votre ||| 1 0.794566 0.73913 0.542431
your |||  your ||| 1 0.896708 0.130435 0.0173818
your ||| .  vos ||| 0.0416667 0.00483387 0.0434783 0.0112108
```

   Each line contains the phrase pairs together with the computed phrase translations scores, in the following order:

   (a) Inverse phrase translation probability $\phi(\bar{e}|\bar{f})$

   (b) Inverse lexical weighting $lex(\bar{e}|\bar{f})$

   (c) Direct phrase translation probability $\phi(\bar{f}|\bar{e})$

   (d) Direct lexical weighting $lex(\bar{f}|\bar{e})$

   The lexical weighting is an additional score to check the reliability of the phrase translation probability [2]. Rare phrase pairs may cause problems if they are collected from noisy data. If both of the phrases $\bar{f}$, $\bar{e}$ only occur once in the training data, then $\phi(\bar{f}|\bar{e}) = \phi(\bar{e}|\bar{f}) = 1$. This often overestimates how reliable rare phrase pairs are. Lexical weighting decomposes phrase pairs into its word translations, so that we can check how well they match up.

5. Build the reordering table. Different reordering model types can be specified[5]. By default, a distance-based reordering model is used. This model gives a cost linear to the reordering distance, e.g. skipping over two words costs twice as much as skipping over one word.

The output of the training process is the phrase translation table, reordering table and a configuration file "moses.ini" containing all necessary configuration for the decoder to translate sentences.

[4] https://github.com/moses-smt/mgiza.git

[5] http://www.statmt.org/moses/?n=FactoredTraining.BuildReorderingModel

6.2.2 Tuning

Tuning refers to the process of finding the optimal weights for the linear translation model of Moses:

$$p(e|f) = \phi(f|e)^{weight\_tm} LM(e)^{weight\_lm} D(e,f)^{weight\_dm} W(e)^{weight\_wp}$$

The probability cost that is assigned to a translation is a product of probability costs of four models:

- $\phi(f|e)$ The **phrase translation table** ensures that the source and target phrases are good translations of each other.
- $LM(e)$ The **language model** ensures that the output of the target language is fluently.
- $D(e,f)$ The **distortion model** allows for reordering of the input sentence, but at a cost. The more reordering, the more expensive is the translation.
- $W(e)$ The **word penalty** ensures that the translations do not get too long or too short.

There is a weight assigned to each of these components that influences their importance. These weights can be passed to the decoder when translating sentences. The optimal weights depend on the languages and training corpus. However, Moses offers to tune the weights automatically by maximizing the BLEU score [5] on a small, separate set of parallel sentences (tuning set).

We used 80% of the available parallel data for training and the remaining 20% for tuning.

6.2.3 Decoding

The decoder of Moses is controlled by a "moses.ini" file. This file stores the path to the phrase translation table and language model and also holds the tuned weights for the translation model. Many more configuration options[6] can be set in this file or passed as parameters when executing the decoder. Here is an overview how the decoder of Moses works, more detailed information can be found in the background section[7]:

1. The decoder collects all phrase translations from the phrase table that could be applied on the input strings of words (translation options). The translation options are stored with the following information:
   - First source word covered
   - Last source word covered
   - Phrase translation in the target language
   - Phrase translation probability
2. The output sentence is generated left to right in form of hypotheses. Each hypothesis is represented by:
   - A link to the best previous state, so that we later can back track through the graph to find the best translation.
   - The cost (probability) and covered source words so far. A low cost means high probability.
   - An estimate of the future cost, how expensive it is to translate the remaining words of the source sentence.

[6] `http://www.statmt.org/moses/?n=Moses.DecoderParameters`
[7] `http://www.statmt.org/moses/?n=Moses.Background`

The decoder uses "Recombination" and a "Beam Search" (see Section 3.1.3) to reduce the size of the search space. The beam search compares hypothesis covering the same number of source words in stacks. Given the cost so far and the future cost estimation, hypotheses that fall outside of the beam are pruned. The beam size can be defined by threshold and histogram pruning. A relative threshold cuts out a hypothesis with a probability less than a factor α of the best hypotheses. Histogram pruning keeps a certain number n of hypotheses in each stack.

6.2.3.1 Example

Let us execute the decoder by translating the same example string as used in the previous chapter "Open the settings to change your username" to French. Here is the command to invoke the decoder:

```
echo 'Open the settings to change your username' | /home/stefan/mosesdecoder/bin/moses
    -f /home/stefan/AppTranslator/data/moses/en-fr/mert-work/moses.ini -verbose 3 &>
    moses_decoding.out
```

We specify the path to the "moses.ini" configuration file and set the verbosity level to 2 (middle) to get some nice debug information.

The decoder produces the following output string: "ouvrir les paramètres pour modifier votre nom d' utilisateur". Here are some statistics on how the decoder derived the target sentence:

- Total translation options: 197
- Total hypotheses considered: 42614
- Total hypotheses discarded: 31220
- Total hypotheses recombined: 8292
- Total hypotheses pruned: 2401

6.3 Tensorflow

7 Web Application

The main purpose of the web application is to offer a simple way of testing the different implemented translation systems (Baseline with Solr, Moses and Tensorflow). However, the application could already be used by app developers to translate their XML translation files.

The interface allows to translate either plain strings or a XML translation file from one language to another. Each decoder offers some specific settings where the user can influence the decoder.

7.1 Architecture

7.2 Frontend

8 Evaluation

8.1 BLEU Score

8.2 T

Bibliography

[1] Philipp Koehn Franz, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. pages 127–133, 2003.

[2] Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 2010.

[3] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-burch, Richard Zens, Marcello Federico, Nicola Bertoldi, Chris Dyer, Brooke Cowan, Wade Shen, Christine Moran, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation, August 12 2015.

[4] Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, 2003.

[5] Kishore Papineni, Salim Roukos, Todd Ward, and Wei jing Zhu. Bleu: a method for automatic evaluation of machine translation. pages 311–318, 2002.

[6] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.