Industrial Strength in Multilingual Analysis
Project Report

Johannes Krämer & Mihai Manolescu

March 2018

Contents

1 Background

With the technological influence of the last decades, our world changed remarkably. The power of Networking enabled more and more ideas to become reality and offer products and services globally.
In order to make successful business around the world, one needs to adjust to the local needs of different cultures and consider traditions, languages into the marketing strategies.
Our project tackles a very specific linguistic problem, which is thought to prevent cultural problems due to the different language use.
For international companies, brands, products today it is mostly tried to find names which are consistently usable around the world. Multiple linguistic problems appear with that desire:

1. should a company-, brand,- product name be translated?
2. which alphabets should be used if there are multiple in the target markets?
3. how does the company-, brand,- product name sound?
   (a) is it pronounceable?
   (b) does it sound strange?
   (c) does it sound like a taboo word?

...to name only a few. We specified on the phonological problem of 3c from the above list. We see in this an expensive problem, in terms of human resources (experts). Our goal was to work in the direction of automating this problem.

2 General information

First it needs to be stated that this project is not based on any previous research. What this project aims for is a explorative basic research towards the above described problem. This means that the results and methods described in this report do not aim for a ready to use tool, but rather for showing tendencies where further research might go.
As shortly described above (Background) this tool aims for checking if potential company-, brand,- product names sound like a taboo word in another language.
An example of this disastrous problem is the car name "e-tron"[1] which

[1] from the company Audi

sounds like "Etron" in french. "Etron" translated to English means "droppings, excrements, ... ". Another example is the car name "MR2"[2] which sound like "Merde" in french. "Merde" translated to English means "shit". More examples can be found in this article. [1]
Since this problem is not related to the writing system of a language alone but also to how a combination of characters is pronounced in other languages, it is important to do the analysis on a deeper level, a level where it is possible to compare the pure sounds with each other: IPA. Going to the level of IPA it enables the comparison of sounds, even when the comparison needs to be done across totally different writing systems like Latin and Chinese (pinyin). With this procedure all kinds of problems emerge [3].

From a users perspective it is possible to pass a potential company-, brand,-product name to the system. The systems answer is an overview for every supported language, if the given word is somehow close to a taboo word in any of those supported languages, and if so, how close. The critical degree of the given word to the closest taboo word for every language is marked by colors from green (indicating no similarity at all) to red (indicating a critical similarity) [4].

3 Linguistic challenges

The first challenge was to make lists with taboo words for each language. This task war relatively easy, because there are lists of for almost every language on Wiktionary [2] and there is a Github project [3] regarding this topic. There weren't any problems with the Latin-written languages. The challenge were the other languages, like Chinese, Japanese, Arabic and Russian. Because the tool provided by Johannes Dellert, which transformed strings into their IPA representation, could only process Latin-written text, these lists with taboo words had to be rewritten in their Latin transcription. Some languages have official transcriptions, like *romaji* for Japanese and *pinyin* for Chinese and we had to manually translate every word from the list to their corresponding transcription. For Russian and Arabic we used the official chat transcription to represent the language in their Latin form. The other challenge was to create a simplified IPA for our program. This had

[2] from the company Toyota
[3] more on this in the section "Linguistic challenges"
[4] the detailed explanation of further UI elements and features will be explained in the "Implementation" part

to be done to make it easier for us to analyze the input word with our taboo lists and to make the program stricter. The idea was to create equivalence classes to represent the IPA table, so that we can use it in our program. We first tried a very strict version, in which the consonants that sounded similar were grouped together to one single consonant. For example the IPA symbols "t, d, ʈ, ɖ, ɟ" were all mapped to the simple letter "T". We quickly realized that it is too general for all languages and the results were not very good. After trying around with different versions of the simplified IPA, we came to the conclusion that we need to make it as broad as possible, so it can cover every language and doesn't change the input word too much. We've changed most of the IPA - consonants to match their corresponding consonants and also added fricatives to our table, which are all matched to the letter "C". For the vowels, we tested out if we can minimize all the vowels to just three, but the results were not satisfying. We therefore included every vowel in our system. There are about 20 equivalence classes that are being used to simplify the IPA and help us compare the input word with our word lists.

4 Project description

4.1 Resources preparation

As a first step, we had to create the lists with the taboo words for every language. Each of us used one of the sources [2] [3] and created text files, which were then merged together using a small script that also checked for duplicates. The second step was to create the simplified IPA, which was created manually. We build the simplified IPA based on the IPA chart and our own knowledge. Another vital step was testing the project and how it worked with words that only had one, two or three letters modified from their original form. These lists were created from the previously created taboo lists. We started by changing one letter at a random position in the word with another random letter. Then did the same and changed two / three letters in a word. All these testing files were created using an algorithm, which randomly replaced the character at a random position in the word.

4.2 Implementation

General

For the implementation the "Google Web Toolkit (GWT)" and "Bootstrap 4" was used. Like in a classical GWT project we have different packages

dividing the backend functionalities, frontend functionalities. There are also some aspects that are shared by both, the backend and the frontend, like the functionality of passing data to each other.

Backend

The backend implementation consists of three logical blogs (packages): the general "server" package, the "distance" package and the "transliteration" package:

1. server:
   The general server package contains only one class, GreetingServiceImpl, which is responsible for creating the results from a high level point of view and communicating them back to the client for showing them to the user.
   In more detail, the above mentioned class is responsible for loading the necessary resources: the lists of taboo words (each language one list), the transcription lists for the non-Latin writing systems, the simplified IPA list. After loading those it distributed the necessary information and resources to the "transliteration" and "distance" packages in order to do the lower level analysis and calculations.
   For returning the results to the client, the results are sorted by the distance criteria, from high to low.

   It also contains some util functionalities for reading files. Due to the requirement of portability and having all the resources within the project itself (instead of having them later in a specific location on the server) it was not possible to outsource the util functionalities in to a different package.

2. distance:
   The distance package contains the following algorithms in form of classes: Levenshtein distance, N-gram distance, Jaro-Winkler distance, longest common substring (in the SimilarityMeasures class). The main focus was on experimenting with the levenshtein distance and the longest common substring algorithms. The others were barely experimented with due to a focus change during the experiments. It turned out that the experiments were not giving the results as planned and expected, which required a even more basic root analysis of the problem, concentrating only on a fixed set of parameters, while creating

more and more test data for scalable tests [5].

There is one more class, the weighted measures where the levenshtein distance and the longest common substring are combined. In the following this combination will be motivated and explained in more detail:

The need for something more then just the levenshtein distance was the fact that the pronunciation of words brings some complex problems with it, which the levenshtein distance is not designed for (since in the original sense it is for similarity calculation of written items). Examples for those problems are the emphasis while pronouncing a word; the location of difference (having two very close sounding words, a change at the beginning of the word is more significant then at the end of the word, causing a bigger difference in the sound similarity); if the two words contain large equal substrings and the change happens only outside this substring the difference in sound can be considered lower then otherwise [6]

An attempt of caching some of the above described problems was to combine the levenshtein distance with the longest common substring. The longest common substring is used in two ways: for search space reduction and for the distance calculation itself.

In the search space reduction only those taboo word candidates are kept for further analysis which have the longest common substring together with the input word [7].

In the combined distance algorithm itself the LCS is used to split the two strings, which are compared against each other, into three parts: the longest common substring itself, the part which comes before the LCS and the part which comes after the LCS[8].

This enables a separate penalty for the front and the end part of the two strings. If for example the front part of one of the words is massively larger then of the other word this is penalized stronger then if the size is the same. In other words disproportionately is heavily pe-

[5] more on that in the "Experiments" section

[6] those criteria are based on the experience of Johannes Dellert and also of our own experience during the development, and can not be confirmed by any literature or research

[7] this actually happens in the GreetingServiceImpl class, but is mentioned here for a consistent explanation of the LCS algorithm

[8] it is possible that in some cases one or more of the parts do not exists: for example when the LCS starts at the beginning of the word or when it goes till the end of the word or when it covers the complete word. This scenario is possible for both strings (which are compared against each other) at the same time or only for one of them, since one word can be a substring of another word

nalized in both cases the beginning of the word and the end of the word.
The distance of the front and end parts of both words are calculated separately with the levenshtein distance. Based on that separate calculation of distance, the front distance is reinforced by an adjustable weight. The additional reinforcement of the front part tackles the problem that a change at the beginning of the word causes a bigger difference in sound then a change at the end of the word, as mentioned above. The different settings of the hyper-parameters which were used during the development and the experiments are explained in the "Experiment" section.

3. transliteration:
   This package contains the following functionalities as classes: a language transliterator, a phonetic transliterator. It also contains a data structure called "source transliteration".
   The language transliterator is responsible for transliterating the input word (given by a user), written in Latin characters, either back to the original writing system or directly to a phonological representation, in order to be able to analyze the input word as a potential non Latin writing system word like Arabic, Russian, etc.. [9]
   The phonetic transliterator is responsible for creating the phonological representation as well as the simplified phonological representation of the taboo words and the input word.[10]

Frontend

The frontend consists only of one package "client", holding the UI elements and functionalities. Additionally there is a "images" package which contains resources (mainly images, but also loading animations) which are used to enhance the UI.
The client package contains the two interfaces needed for the proxy to work. It also contains the ISMLAProject class which is the entry point (the root widget of the UI is instantiated there and added to the DOM).
The main two classes are the "TokenTabooFinderUiBinder" and the "LanguageCard" which are both self created widget for the needs of the frontend

[9] depending on the language and the available resources and external tools for that language

[10] the non-simplified phonological representation is mainly achieved by using an external dependency provided by Johannes Dellert

part.
The "TokenTabooFinderUiBinder" is responsible for displaying the heading of the tool together with an input area where the user can type in his company-, brand,- product name.
The "LanguageCard" is added dynamically after the distance calculations. Each language card corresponds to one language. The language card is divided into three sections.

1. shows the language at the top, with a representative country flag and also the name of the language.
2. shows the closest taboo word, the translation (if available in the resources), and links to Google translate and Wiktionary searching for the particular taboo word. The links are motivated by the finite amount of resources for translations in this project.
3. marks the degree of appropriateness by colors from red to green, together with a word stating the same

4.3 Experiments

One of the main parts of our project was the creation of an initial test set and testing the different distance algorithms to see which configuration worked best. The motivation for doing such experiments was, that we didn't have any given test set with which we could test our program. This caused trouble during the evaluation of the program. It turned out very fast that it is not feasible to judge our algorithm based on a couple of examples that we tried by hand. At this moment the need for a scalable test set emerged. First we came up with the idea of creating random three syllable words with the pattern 'CVCVCV'. The test file contained over 100.000 words and we chose 100 random words out of that list to test our algorithm. The results were not very informative. There were 2-3 words close to one taboo word in all the lists. The problem was again, that it was not really possible to detect if the results were nearly good. As we will see later, the algorithm worked correctly in that respect, that he did not find any connection to taboo words - since there were none. But from those results it was not possible to tell whether the algorithm would detect critical similarities. After a closer manual look on the 'CVCVCV' words, we realized that random generated three syllable words are too broad and there aren't really matching taboo words in any language. There are only a few languages that have such a pattern as our three syllable words, e.g. Spanish, and not even these languages showed good results. After more tests, we realized that these test files don't really

help us improve our program. What we needed was a test set from which we could tell, if words, from which we know that they are close to taboo words, are also recognized as such. The test set should be scalable as well and it should be possible to create it in a reasonable amount of time, without putting the whole project time into manual work. Therefore we had the idea of modifying our existing lists of taboo words.
At start, we created seven test files for each language. These files had the taboo words inside, but these were slightly modified. At the beginning, one random character at a random position in a word was replaced by a random consonant. We proceeded with this method and replaced one random character at the beginning of the word and did the same at the end of the word. After these three test files were created, we created files where two characters were replaced in a word and also one file where there were three characters replaced. There were four different tests that we did with distinct weighting on the different parts of the word. Using LCS or not before weighing was one part of the test, since without LCS we are comparing with all taboo words and with LCS we are only comparing the taboo words that share the same longest common substring. So LCS is used as a search space reduction in this case. The second part, where LCS was used, was for splitting the actual word to be tested and a taboo word candidate into three parts: the actual longest common substring itself, the part before and the part after it. Then testing out if weighing the part before the LCS makes the results better. We tested it with weighing the part before the LCS twice or not at all, while the part after the LCS wasn't modified. The last configuration was without weighing the different parts of the word. After this first testing phase, we weren't very satisfied with the results. The best configuration out of these four, was the one using LCS and the weight on the part before the LCS was weighted twice. The tests didn't show that these words were close to the original, not even the words that only had one character replaced. After this testing phase, we changed the weight of the different sections of word slightly and tested again. Results were a bit better, but there were still very few words recognized by our system for the languages. The weights still weren't perfect and we changed them a few times until we were satisfied. We also realized that our algorithm sometimes matched the modified words with very long taboo words, due to a long LCS, which was not what we wanted. Therefore we changed the algorithm to punish these different word lengths, so that words with similar word length are being matched. After changing the distance algorithm another time, results finally got better. There was one file for each language now, where the words had one random character modified at a random position. We tested the system with each of the Latin-

written European languages, e.g. German, Italian, Spanish, etc., because it was the easiest way to check how good out program works. The results were finally very good and for almost every word in the test file the corresponding taboo word was matched for the given language. Because of the similarity of the languages, there were more matches for one word. This showed us that our algorithm and the weights on the different parts of the word finally worked well. From 175 words in our German test file, about 160 were recognized by the system, which meant that our algorithm was accurate and could recognize the corresponding taboo words. It is important to point out, that we achieved good results in a very restricted context. We had to break down our expectations and goals many times and focus on more basic and rudimentary functionality.

4.4 Work splitting

We split the work into four different parts. The resources were mostly put together by Mihai, Johannes provided the helper scripts that were used for the few cases where algorithmic preparation of the data was possible. The backend implementation was mostly done by Johannes. The frontend was designed and written by both of us. Testing was mostly done by Mihai, but also by Johannes, who also came up with the idea of the new testing principle, creating the slightly modified taboo words files.

5 Further research

In the end there are even more questions and problems to solve than before, but the problems get more concrete:
Further research could be faced with the investigation of word comparison at a phonetic level. We have one single general IPA transformation list for all the languages. Probably this is still to broad, and a more detailed investigation is necessary of how words of one language are pronounced in another language. Such insights could help creating even multiple IPA transformations lists for the simplified IPA for different languages.
Since we focused on detecting very similar words, it is important to tackle the problem of granularity when the similarity gets blurry but is still there. The creation of a proper test set is still an open point. We tried our own ideas, but surely this is not representative for the real use of potential new company-, brand,- product names.
Also we worked with a very limited amount of algorithms, a broader landscape of algorithms should be tested and applied to this problem, presupposing a proper test set is available.

6 Conclusion

Based on the beforehand mentioned algorithms, our project is set to be the first step in creating a tool for international companies to use to check their product names in different languages. This topic is also important in the future, as companies are expanding worldwide and are trying to make a name for themselves.
As already mentioned this project is highly experimental and not a ready to use tool. Some of the challenges were to come up with a realistic test set, introducing hyper-parameters in order to hit the main aspects of a word comparison, bringing the words to the simplified IPA level and many more.

References

Online sources

[1] URL: https://www.welt.de/motor/article13669067/Diese-Autonamen-wurden-zu-PR-Totalschaeden.html (visited on 03/15/2018) (cit. on p. 3).

[2] URL: https://en.wiktionary.org/wiki/Category:Vulgarities_by_language (visited on 03/15/2018) (cit. on pp. 3, 4).

[3] URL: https://github.com/LDNOOBW/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words (visited on 03/15/2018) (cit. on pp. 3, 4).