Identifying Sound Correspondence Rules in Bilingual Wordlists

Blaschke Verena, Korniyenko Maxim

University of Tübingen

1. Introduction

Historical linguistics aim to solve many different linguistic problems. Identifying regular, systematic sound correspondences between the vocabularies of two or more languages is one of them. Given a large set of such correspondences, it can be surmised that the languages share a common ancestor.

Different linguists already made great progress in this area. To solve this problem, Kondrack, 2003 [1] and Rama et al., 2013 [2] use pointwise mutual information. Another possible solution is analyzed by Hoste et al., 2004 [3] who compare rule sequence learners and decision trees. Wetting et al., 2012 [4] in their work introduced a decision-tree algorithm that uses (left-hand) context information and phonological features. The last paper inspired us to perform our own experiment.

In our work, we aim to establish sound correspondences in several steps. First, we identify potential cognates–words that share a common origin (cognates often have retained similar meanings, but not always). To solve this problem, we use Needleman-Wunsch alignment and Normalized Edit Distance. The next step is to establish regular correspondences between the potential cognates using Decision Trees.

Our paper contains 6 sections. In section 2, we analyze the data, its representation and preprocessing. In this section we also cover the alignment and detection of possible cognate pairs. In section 3, we go though the process of creating tree classifier and preparing the data for it. Rule extraction is described in section 4. We introduce our evaluation approach in section 5. A small discussion concerning several aspects of our work can be found in the last section.

2. Preprocessing

2.1. Data

We use NorthEuraLex [5], a database containing lists of over a thousand concepts across over a hundred (mostly) North Eurasian languages. From this database, we extracted the concept lists for Ukrainian, Russian, Swedish, and German[1]. We use these lists to extract pairs of potential Russian-Ukrainian and German-Swedish cognate pairs, as detailed in section 2.3.

In order to prepare the raw data for further processing, we merge the list of each language pair (deu-swe, rus-ukr) using the script `preprocessing/merge-lists.py`. When merging the word lists, we take care to align them by concept, because some concepts can have more than one representation. The merged lists are stored in the folder `data` with names of the pattern <*language 1*>`-`<*language 2*>`-all.csv`.

[1]Downloaded from `http://northeuralex.org/languages/`<*language ISO code*> on November 21, 2017 (version 0.9).

2.2. Sound representation

In our experiment, each sound is represented as an object of class `Phone`. For describing phones, we decided to follow the approach that was introduced by Wettig et al., 2012 [4]. As in the original paper, each sound contains the information about its **Type**, consonantal information (**Manner**, **Place**, **Voiced** and **Secondary**) and mostly vocalic information (**Vertical**, **Horizontal**, **Rounding**, **Length**). We also added **Nasalization** as a separate feature[2]. Such a detailed form of phone representation gives better results both for cognate detection and rule production. For more details, see Table 1. It is also worth mentioning that some features can be equal to zero ("not applicable"). For example, any vowel would have consonantal features equal to zero, and vice versa.

Feature	**Possible values**
Sound Type	dot, word boundary, consonant, vowel
Manner	plosive, tap, trill, affricate, fricative, etc.
Place	glottal, pharyngeal, uvular, velar, etc.
Voice	voiceless, voiced
Secondary	pharyngealized, velarized, palatalized, etc.
Length	normal, half-long, long
Vertical	open, near-open, open-mid, mid, etc.
Horizontal	back, near-back, central, near-frond, etc.
Rounding	unrounded, rounded
Nasalization	nasalized

Table 1: Features of Phone objects

Last but not least is the method `distance` of class `Phone` that is used to calculate the distance between two objects of this class. It returns a float between 0 and 1, where 0 signifies equality and 1 is the maximum distance. When calculating the distance, we compare the values of two `Phones`' features. If the `Phones` are of different type, we directly return 0, otherwise we return the average distance between the type-specific feature values. The distances are again mostly binary (0 for equal values, 1 for different ones). If a feature is scalar (like, e.g., the vertical position for a vowel), we penalize different values that are near each other on the scale (like, e.g., *open* and *open-mid*) with only 0.5.

In order to transform IPA symbols into objects of class `Phone`, we created a dictionary that contains IPA symbols and corresponding phonological features (`ipa_dictionary`). All the features were taken from the official IPA chart [6]. The file `data/ipa.csv` contains the features as strings. We use `preprocessing/transform_ipa.py` to create `data/ipa_numerical.csv`, which contains the same fea-

[2]We originally considered also analyzing Romance language pairs, including French, for which nasalization is a relevant feature. However, because of time constraints, as this work is a term project, we decided to ultimately only focus on the previously mentioned two language pairs.

tures encoded as integers.[3] Given the `ipa_dictionary` one can transform any IPA symbol to `Phone` object using `to_phone` method in `preprocessing/utils.py`.

2.3. Determining pairs of cognates

First, we apply the Needleman-Wunsch algorithm[7] to align word pairs. In order to align two sequences *f* and *g* of length *n* and *m* respectively, the algorithm computes the optimal alignment in four steps.:

1. create a score matrix *A* and a trace matrix *B*, both of size $(m + 1) \times (n + 1)$
2. initialize the matrices (detailed below)
3. compute the values for the matrices (detailed below)
4. trace back the optimal alignment

In both matrices each segment of *f* is confronted with the corresponding segment of *g*. The value in A_{ij} represents the score for the best alignment between the two substrings of *f* and *g*, which end at segment *i* and *j* respectively. The initialization process starts with setting a zero in the cell A_{00}. Then the first row and the first column are filled with multiples of the gap score. In our case we set the gap score to the common value of -1. As a result, the first column represents the alignment of *f* with a sequence of gaps, and the first row an alignment of *g* with a sequence of gaps. As long as we have the values for the first row and the first column, we can calculate the values for the rest of the cells using equation 1.

$$A_{ij} = max \begin{cases} if\, phone\_dist < 0.5 & A_{i-1j-1} + 2 - phone\_dist \\ if\, phone\_dist \geq 0.5 & A_{i-1j-1} - 1 \\ A_{i-1j} + gap\_score \\ A_{ij-1} + gap\_score \end{cases} \tag{1}$$

We slightly modified the vanilla Needleman-Wunsch algorithm. Instead of using a simple match/mismatch choice, we use the `phone_dist` that was presented in the previous section. Phones with distance that is less than 0.5 are treated as a match case, and phones with higher distances as a mismatch. This allows us to find sound pairs more accurately. For instance, vanilla Needleman-Wunsch would consider the sound [k] and its palatalized version [k^j] to be totally different sounds and compute a mismatch case. With `phone_dist`, we can avoid such situations.

During the process of computation the values for matrices, it is necessary to keep track of the origin of the value that is assigned to each cell. Such information is saved in matrix B.

When both of the tables are finished, we can *traceback* the optimal alignment. Starting from the bottom-right cell, we move to the top-left cell using information from both matrices. We can only move to adjacent cells to the left, top, or top-left diagonal. Our goal is to maximize the similarity, so the trace with the highest score represents the optimal alignment. Based on the movement, two symbols are either matched, or aligned with a gap: In case of a movement from A_{ij} to A_{j-1i-1}, f_i gets aligned with g_i. If the movement occurs to $A_i - 1j$, f_i gets aligned to a gap and for $Aij - 1$ and g_i respectively. We represent each gap with an asterisk (later, and in [4], also called "dot symbol"). We also prefix a hash (#) to each word, signifying the left-hand word boundary. Our implementation of the Needleman-Wunsch algorithm can be found in `preprocessing/utils.py`.

Finally, we attempt to only use word pairs that are cognates and discard those that are not. For this purpose we use the modified Normalized Edit Distance that takes into account the phonetic distance of `Phone`. The following method can also be found in `preprocessing/utils.py`. After aligning the word pairs as mentioned above, we calculate their Levenshtein edit distance (with an insertion/deletion cost of 1 and a (mis)match cost given by the `Phone` distance). After applying the method, cognates and non-cognates are stored as a *<language 1>*`-`*<language 2>*`-cognates.csv` and *<language 1>*`-`*<language 2>*`-non-cognates.csv` respectively in `data`.

3. Feature extraction and building decision trees

We take 90% of the cognate pairs to produce the training data for our trees. The rest is left for evaluation, which is described in Section 5.

The training data is represented as a matrix of size $n \times m$, where n is a number of sound pairs and m is the number of features that is equal to $num\_levels \times num\_positions \times num\_features$. Since we work with bilingual sound pairs, we have two **levels** (*source* and *target*). **Position** describes the information that is relative to the position currently being coded. There are eight positions that describe the *phone itself*, *previous symbol*, *previous non-dot symbol*, *previous consonant*, *previous vowel*, *previous or self non-dot symbol*, *previous or self consonant*, and *previous or self vowel*. The `Phone` from each position has ten phonological **features**, as described in section 2.2. As a result, we have $m = 2\ levels \times 8\ positions \times 10\ phonological\ features = 160$; that is, 160 features per aligned symbol.

Let us have a look at one example. Given an aligned word pair such as ['#', 'h', 'a', 'l', 's'] and ['#', 'h', 'a', 'l:', 's'], there will be four sound pairs (h-h, a-a, l-l:, s-s) (following the redundant alignment #-#). The first sound of the pair is in the source position while a second one is in the target position. The matrix for such data would be of size 4×160 with 4 training instances and 160 features.

We extract the features in `preprocessing/features.py`, making use of `preprocessing/candidate_contexts.py`, and save them as `data/`*<language 1>*`-`*<language 2>*`-features.csv`.

Before feeding the features to the decision tree classifier, we need to remove some columns. First, we need to extract the column of the feature that we are looking for as a label. Second, we need to exclude features that are directly connected to the sound itself. For this purpose, we remove all columns that have *phone itself*, *prev or self non-dot symbol*, *prev or self consonant*, *prev or self vowel* as their positions if they have the same level as the sound whose features we want to predict. We then build the decision trees using the `Decision Tree Classifier`[4] from the Python library sklearn. For our trees we were setting min_samples_leaf to 0.01. Increasing this pa-

[3]Such a representation can also be considered as a vector.

[4]The official documentation for sklearn.tree.DecisionTreeClassifier can be found at `http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`

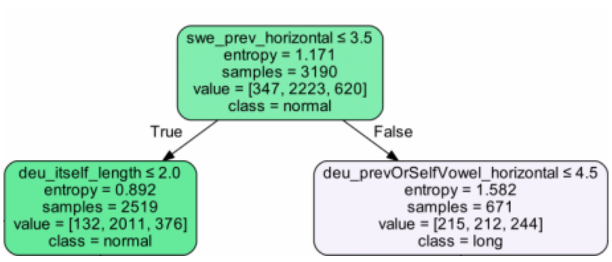

Figure 1: Part of the decision tree that predicts sound length for Swedish visualized with GraphViz. The first line in each node gives the decision that is used for splitting the dataset. The last line describes the majority feature of the samples described by the node.

rameter decreases the size of trees but evaluation results also decrease. Once the tree is trained, we save it for evaluation under `evaluation/classifiers` and additionally save a visualization that we create via `GraphViz`[5]. Figure 1 gives an example. Altogether, for one pair of languages, we need to create 18 trees (9 trees for each phonological feature of a language except **sound type**).

The methods that are responsible for creating trees can be found in `tree/tree.py`

4. Rule extraction

Scikit-learn's decision tree classifier currently[6] does not support pruning or removing redundant sections. However, we can remove such redundancies while transforming a tree into a set of conditional rules.

To extract the rules, we first perform a pre-order tree traversal during which we keep track of the decision nodes that we come across. Whenever we encounter a leaf node[7], we save information about the decision nodes leading to the leaf node and the category that is predicted by the leaf node as a rule. We describe each decision node as a combination of the feature it is about, the numerical threshold it uses for splitting the training data set, and the decision that leads to the leaf node. The decision is `True` if the queried element's value for the given feature is below the threshold, `False` otherwise. Each rule then consists of three lists (features, thresholds, decisions) and the predicted category. The lists can be considered to be parallel in that they describe the same decision nodes in the same order.

After extracting the rules, we remove redundancies from individual rules and the rule set as a whole:

First, if possible, we prune individual rules by removing superfluous decision nodes. If two or more decision nodes belonging to a rule describe the same feature and contain the same decision values (all `True` or all `False`), we shorten that rule by removing the redundant nodes. For instance, if a rule contains two nodes about a shared feature, both of which have the decision `True` (indicating that the feature value is *below* both thresholds), then it is sufficient to only use the decision node with the lowest threshold.

Second, we merge pairs of rules that, as a pair, contain redundant decisions. This is the case whenever there are two rules that predict the same class and describe identical sets of decision nodes, except for one node that only differs in its decision value. In that case, this one decision node is irrelevant for predicting the output. We therefore merge the two rules, excluding the decision node in question. We repeat this second step for the pruned rule set until no more rules can be merged.

Finally, we convert the rules into an easy-to-understand format by describing the individual decision nodes as set membership (`IF value in {X, Y, Z}`) or identity (`IF value is X`) conditions. Sets that contain all possible values for a feature except for "not applicable" (`N/A`) are summarized in the format `IF feature applies`. We then concatenate the stringified rule segments and add information on the output that they predict. Figure 2 gives an example. Note that if a decision tree only predicts a single category across all of its leaf nodes (like, e.g, the tree for nasalization in Russian), the corresponding rule set is a singleton containing only the consequent of the rule.

```
IF swe_itself_sound_type in consonant,
vowel AND swe_itself_horizontal is N/A
AND swe_prevOrSelfCons_manner is lateral
approximant THEN lateral approximant

IF swe_itself_manner is nasal AND
swe_itself_horizontal is N/A AND
swe_prevOrSelfCons_manner in approximant,
nasal THEN nasal
```

Figure 2: Excerpt from `output/deu_manner_rules.txt`

We save the rule sets in the `output` folder. The rule extraction and cleaning is performed in `tree/rules.py` and tested in `test/ruletest.py`.

5. Evaluation

For evaluating of our program, we decided to perform the imputation of sounds using the decision trees that were created. As test data, we use 10% from the original, already aligned cognate pairs. We use the whole source word but the target word should be generated by the trees. While we perform the first step of the prediction, we have all the information about the source sound and the knowledge that the predicted sound of the target level is preceded by the word boundary symbol. Based on this data and using the trees, we predict the nine phonological features of the target sound. Depending on predicted features, we determine the **sound type** of the symbol. The information about the predicted sound is used in the process of predicting next sound and so on till the end of the word. Because the words were already aligned, we know that the target word contains as many symbols as the source word.

Once the entire target word is predicted, we compare it with the actual target word from the test data. We use the Normalized Edit Distance from the preprocessing step that was described in section 2.3. The results can be found in Table 5.

The code for evaluation can be found in `evaluation/evaluation.py` Results of the evaluation can be found in files *<source_language>*`-`*<target_language>*`-evaluation.csv` in `output`.

[5]Official web-page: `http://www.graphviz.org`

[6]As of version 0.19.1.

[7]A tree node which does not induce a split of the dataset, but instead gives the label that is predicted for data points that end up in this node.

Target Language	Source Language	Average NLD
German	Swedish	0.11
Swedish	German	0.13
Russian	Ukrainian	0.16
Ukrainian	Russian	0.07

Table 2: Average NED for imputed words

6. Discussion and Conclusions

6.1. Data

We used NorthEuraLex as data source for our experiment, and it proved very useful for our purposes, albeit not perfect. The phonological information there is largely automatically generated, which means that some entries contain erroneous information. Furthermore, the transcription in this database is somewhere between phonetic and phonemic. This means that some information is missing (for example, aspiration in German). The creators of the NorthEuraLex mention that they omitted some "distinctions that are not of central importance to historical linguistics" [8]. It still would be a interesting to run the experiment using the data from different sources.

For the experiment, we examined two language pairs (German–Swedish, Ukrainian–Russian). It would be interesting to run it on different language pairs (even from different subfamilies) and to compare results.

6.2. Sound representation

Class `Phone` has a variety of phonological features but it does not represent the whole diversity of phonetics. It could be improved with other features (tone, airstream, stress, etc.).

6.3. Determining pairs of cognates

We did not implement the steps for word re-alignment and rebuilding trees that were used in the original paper [4], because of the complex alignment process for determining cognates and because of the scope of this project as a term project. It would nevertheless be interesting for future research to experiment with the re-alignment and tree re-building process, both with our phonetically motivated edit distance measure and vanilla Needleman-Wunsch/Levenshtein.

6.4. Feature extraction and building decision trees

The positional features only capture a symbol's left-hand context. In future projects, including the right-hand context could improve the results by modelling more phonological processes.

6.5. Evaluation

On the evaluation step, we calculate the average normalized edit distance for generated words and target words. It would be a insightful to also print the generated words. We did not implement this because some of the sounds can have a combination of features that does not correspond to any of IPA sound. Taking the closest sound to the generated one could be a solution for potential future work. It would help to make the analysis of our results more useful.

Finally, performing literature research to compile sets of rules for a recall-like measure or to evaluate the validity of the generated rules would be very informative. Unfortunately, this proved to be outside the scope for this term project.

7. Bibliography

[1] G. Kondrak, "Identifying Complex Sound Correspondences in Bilingual Wordlists," in *International Conference on Intelligent Text Processing and Computational Linguistics*. Springer, 2003, pp. 432–443.

[2] T. Rama, P. Kolachina, and S. Kolachina, "Two Methods for Automatic Identification of Cognates," in *Proceedings of the 5th QITL Conference*, 2013, pp. 76–80.

[3] V. Hoste, W. Daelemans, and S. Gillis, "Using Rule-induction Techniques to Model Pronunciation Variation in Dutch," *Computer Speech & Language*, vol. 18, no. 1, pp. 1–23, 2004.

[4] H. Wettig, K. Reshetnikov, and R. Yangarber, "Using Context and Phonetic Features in Models of Etymological Sound Change," in *Proceedings of the EACL 2012 Joint Workshop of LINGVIS & UNCLH*. Association for Computational Linguistics, 2012, pp. 108–116.

[5] J. Dellert and G. Jäger, "NorthEuraLex (version 0.9)," 2017.

[6] International Phonetic Association, "IPA chart," 2015, https://www.internationalphoneticassociation.org/content/ipa-chart.

[7] S. B. Needleman and C. D. Wunsch, "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins," *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[8] J. Dellert, "Compiling the Uralic Dataset for NorthEuraLex, a Lexicostatistical Database of Northern Eurasia," in *Septentrio Conference Series*, no. 2, 2015, pp. 34–44.